www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

DevMedia
Ano 7 - Edição 89 - R$9,90

Agora com + páginas e com duas grandes novidades!

- Artigos de PHP
- e muito mais artigos para iniciantes em Delphi

DELPHI PARA INICIANTES

Aprenda a criar sistemas de cadastro passo a passo

Na Web >> Confira no portal ClubeDelphi PLUS 3 video-aulas sobre como usar Delphi com Excel

Mini-curso
Construa um sistema completo de Contas a Pagar e Cobrança com dbExpress e Firebird 2.0 – Parte 3

Mobilidade
Desenvolva aplicativos móveis com Delphi e Pocket PCs

ASP.NET
Crie menus profissionais para suas aplicações Web

Mão na Massa
Crie um fórum de discussão com ASP.NET

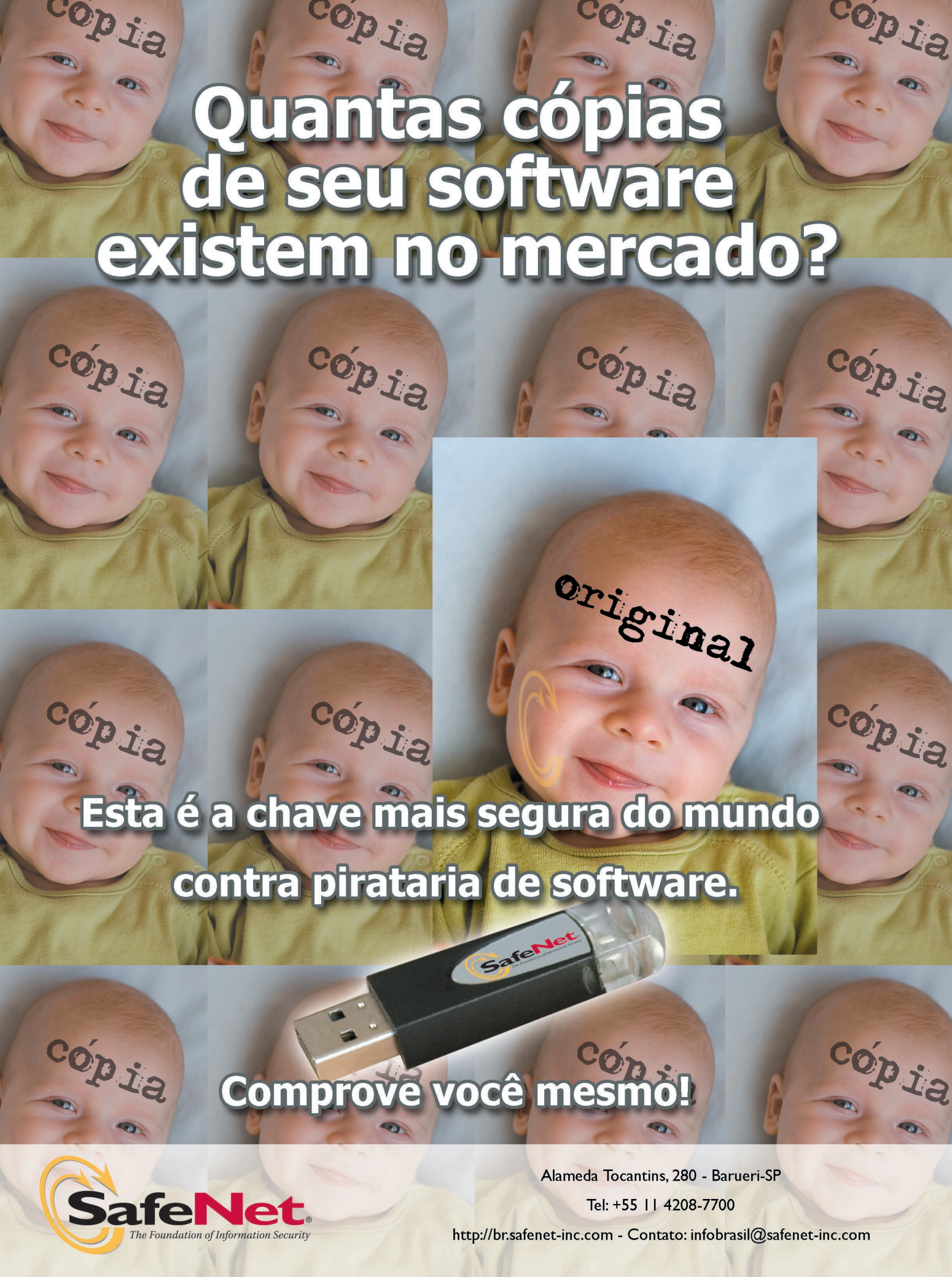
PHP
Mini-curso de Delphi for PHP – Crie uma aplicação Web completa – Parte 2

Easy Delphi
Reaproveite formulários prontos usando herança visual

PHP
Mini-Curso - Introdução à linguagem PHP

Coluna Ask the Expert

Quantas cópias de seu software existem no mercado?



Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 7 - 89ª Edição - 2007 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Equipe Editorial
Adriano Santos, Bruno Lichot, Everson Volaco, Fabrício Desbessel, Gustavo Chaurais e Michael Benford

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
antonioxavier@devmedia.com.br

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@devmedia.com.br

Revisão
Rafael de Castro Santos
rafael@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse: *http://www.devmedia.com.br/clubedelphi/portal.asp*

Apoio

ASSESPRO PARANÁ

SEÇÃO DELPHI: Artigos intermediários sobre Delphi, Win32 e Delphi .net.

06 - PocketPC com Delphi
Manoel Campos da Silva Filho

18 - Contas a Pagar e Cobrança
Adriano Santos

24 - skmMenu
Luís Fernando Quitaiski

30 - Crie um fórum de discussão
Maikel Marcelo Scheid

SEÇÃO EASY DELPHI: Artigos para iniciantes em Delphi

40 - Cadastros simples
Adriano Santos

48 - Herança de formulários
Fabrício Desbessel

SEÇÃO PHP: Artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

54 - Introdução ao PHP – Parte 1
Ewald Geschwinde e Hans-Juergen Schoenig

62 - Delphi for PHP
Fabrício Desbessel

EDITORIAL

Há alguns meses, realizamos uma pesquisa de opinião junto aos nossos leitores. Gostaríamos de agradecer desde já o carinho de todos, suas sugestões foram muito importantes para que pudéssemos traçar os rumos da ClubeDelphi para o futuro. E o resultado você já pode conferir a partir desta edição! A Revista está maior, e com duas novas grandes colunas: Easy Delphi e PHP.

A Easy Delphi é destinada aos leitores que possuem pouca experiência com o desenvolvimento Delphi, mas desejam se aprofundar na ferramenta. Para começar, o Adriano Santos mostra como criar esta que é provavelmente a aplicação mais simples e cotidiana, um cadastro. E se for para começar, por que não começar da maneira certa? Faça o cadastro com o ClientDataSet, componente que o acompanhará provavelmente por toda a sua jornada com o Delphi. Outro recurso que sem dúvida qualquer um vai utilizar, e se for para saber que saiba que ele existe desde o início, é a herança visual de formulários, apresentada no artigo de Fabrício Desbessel.

Falando em PHP, temos dois artigos. No artigo de Ewald Geschwinde e Hans-Juergen Schoenig aprenda os fundamentos da linguagem, para aplicar no Delphi for PHP. No artigo de Fabrício Desbessel, veja a continuação do curso que está arrasando: Criando aplicações na prática com Delphi for PHP. Nesta parte do curso, aprenda como utilizar com o PHP a tecnologia que já revolucionou o conceito de desenvolvimento Web, o AJAX.

E temos mais sobre desenvolvimento Web. O Luís Fernando apresenta um componente pronto para criar menus em ASP.NET, baseados em arquivos XML de configuração. Ainda em ASP.NET, veja como criar com o Delphi uma das aplicações mais populares da Web, fóruns, no artigo do Maikel.

O Manoel Campos desvenda do início ao fim o desenvolvimento de aplicativos móveis para Pocket PCs com Delphi. E para finalizar, o Adriano traz mais uma parte do seu curso de Contas a Pagar e Cobrança, que utiliza os principais conceitos do desenvolvimento com o Delphi hoje: Delphi Win32, Client/Server, dbExpress e Firebird.

Sucesso a todos, muito Delphi na cabeça e boa leitura!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@devmedia.com.br

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 460 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as informações abaixo: **Login: DVM.CD e Senha: SES03**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Luciano Pimenta**
(envie as suas para lucianopimenta@devmedia.com.br)

Como calcular o tamanho de um diretório

Preciso saber qual o tamanho de um diretório da minha aplicação onde salvo fotos, não quero que fique muito grande. É possível fazer isso?

Jefferson Dias

Olá Jefferson, use a seguinte função (**Listagem 1**), que procura por arquivos normais, além de ocultos, de sistema e compactados. Usa um algoritmo recursivo para procurar em todos os subdiretórios. Apenas forneça um diretório inicial como parâmetro.

Veja na **Figura 1**, como ficou o exemplo quando verificamos o tamanho do diretório *C:\Windows\System32*.

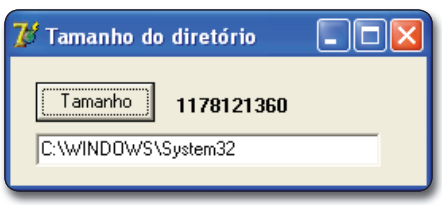


Figura 1. Verificando o tamanho do diretório

Listagem 1. Exemplo de função de busca

```
var
  DirBytes: integer;

function FolderSize(Dir: string): integer;
var
  SearchRec: TSearchRec;
  Separator: string;
begin
  Result := 0;
  if Copy(Dir, Length(Dir), 1) = '\' then
    Separator := ''
  else
    Separator := '\';
  if FindFirst(Dir + Separator + '*.*', faAnyFile,
    SearchRec) = 0 then
  begin
    if FileExists(Dir + Separator +
      SearchRec.Name) then
      DirBytes := DirBytes + SearchRec.Size
    else if DirectoryExists(Dir + Separator +
      SearchRec.Name) then
    begin
      if (SearchRec.Name <> '.') and
        (SearchRec.Name <> '..') then
        FolderSize(Dir + Separator + SearchRec.Name);
    end;
    while FindNext(SearchRec) = 0 do
    begin
      if FileExists(Dir + Separator +
        SearchRec.Name) then
        DirBytes := DirBytes + SearchRec.Size
      else if DirectoryExists(Dir + Separator +
        SearchRec.Name) then
      begin
        if (SearchRec.Name <> '.') and
          (SearchRec.Name <> '..') then
          FolderSize(Dir + Separator +
            SearchRec.Name);
      end;
    end;
  end;
  FindClose(SearchRec);
  Result := DirBytes;
end;
```

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

+460 vídeo aulas | 6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 460 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;
Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).
Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.
Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Curso Online - PLUS

O portal ClubeDelphi PLUS conta com 6 cursos online, confira!

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [*Luciano Pimenta*]

Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [*Guinther Pauli*]

Aplicações client/server no Delphi 2006 [*Luciano Pimenta*]

Criando uma Aplicação multi-camadas Completa com Delphi [*Guinther Pauli*]

Criando uma aplicação completa: sistema SysCash [*Everson Volaco*]

Aplicações Client/Server com dbExpress e Firebird [*Luciano Pimenta*]

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS 3 vídeo aulas sobre como usar Delphi com Excel

Delphi com Excel
www.devmedia.com.br/articles/listcomp.asp?txtsearch=Delphi+com+Excel

Últimas Vídeo-Aulas (Somente para assinantes)

Sistema completo com Delphi 7, dbExpress e Firebird 2.0
Acompanhe a finalização do sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta, mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a criação de usuários do sistema.

Segredos do DBGrid - Parte V a VI
Veja nessas vídeo aulas de Guinther Pauli, várias dicas avançadas de uns dos componentes mais usados da VCL, o DBGrid.

Descompactando arquivos RAR com Delphi – Parte I a V
Veja nessas vídeo aulas de Paulo Quicoli, como utilizar o Delphi para descompactar/compactar arquivos do tipo RAR.

Desenvolvendo uma aplicação para Palm - Parte I a III
Veja nessas vídeo aulas de Ricardo Boaro, como desenvolver aplicação Palm/OS com PocketStudio.

Crie um loja virtual com Delphi for PHP – Parte II
Veja nessa vídeo aula de Fabricio Desbessel, a continuação de mais um curso on-line do Portal ClubeDelphi. Aprenda a criar uma loja virtual com Delphi for PHP.

Conhecendo o Quick Report (Easy Delphi)
Veja nessa vídeo aula de Adriano Santos, as caracteristicas de um dos geradores de relatórios mais utilizados em aplicações Delphi, o Quick Report.

Trabalhando com DLLs no Delphi 2006
Veja na vídeo aula de Ricardo Boaro, como trabalhar com DLLs no Delphi 2006.

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

PocketPC com Delphi

Desenvolvendo aplicativos móveis – Parte 1

Manoel Campos da Silva Filho
(manoelcampos@gmail.com)
é Analista de Sistemas, Tecnólogo em Processamento de Dados pela Universidade do Tocantins, especializando em Gestão e Consultoria em Telecomunicações, Professor da Escola Técnica Federal de Palmas-TO. Trabalha com desenvolvimento em Delphi, PHP e ASP.NET, programação para Linux em C. Desenvolve em Delphi desde a versão 3.0

Dispositivos móveis como PocketPC e SmartPhones têm se popularizado bastante. Os dispositivos possuem atualmente um hardware mais robusto, com mais memória, maior capacidade de processamento e um sistema operacional mais moderno que possui muitas das funções existentes em um computador desktop.

Atualmente dispositivos como o PocketPC, são bastante utilizados para substituir formulários de papel utilizados por empresas que possuem funcionários externos, fazendo coleta de dados diretamente no dispositivo. Uma aplicação muito prática desses dispositivos é em pesquisas de opinião, onde convencionalmente pode-se utilizar formulários de papel que depois devem ter seus dados digitados ao chegar à empresa.

Com o PocketPC, o funcionário registra os questionários direto no dispositivo. Esses dados podem ser armazenados no dispositivo ou enviados diretamente a um servidor de banco de dados. Porém, como nem sempre há conectividade entre o dispositivo móvel e um servidor, os dados são gravados localmente e enviados posteriormente ao servidor.

Neste artigo, veremos os recursos necessários para desenvolver aplicações para dispositivos móveis no BDS2006 e construiremos uma aplicação de pesquisa de opinião com acesso a um banco de dados PostgreSQL. Você pode utilizar qualquer banco de dados que desejar, como por exemplo, o Firebird ou MySQL.

Vamos configurar o ambiente de desenvolvimento, aprender a utilizar o Compact Framework Build Helper e os emuladores, configurações de rede e outras opções.

.NET Compact Framework

O .NET Compact Framework (.NET CF) é um subconjunto do .NET Framework para desenvolvimento de aplicações para dispositivos móveis. Como os PocketPC's

e SmartPhones têm restrições de tamanho de tela, capacidade de processamento e memória, não temos disponíveis todas as classes e componentes existentes no .NET Framework.

O BDS2006 possui um compilador para .NET CF, mas o *Form Designer* não está adaptado para construir formulários para esse Framework, que não possui algumas propriedades e métodos que existem para aplicações Windows Forms para computadores desktop com o .NET Framework completo.

Download das ferramentas

Para montar o ambiente de desenvolvimento será necessário fazer download de algumas ferramentas. Algumas dessas requerem uma conexão com a internet durante a instalação, pois elas precisam baixar pacotes adicionais.

Outras avisarão, durante a instalação, sobre a falta do Microsoft Embedded Visual C++. Essa ferramenta não é necessária, pois não desenvolveremos aplicações em C++. A seguir, temos a lista de ferramentas a serem baixadas:

- *.NET Compact Framework 1.0 Service Pack 3 BR Redistributable*: Pacote contendo os *assemblies* para rodar aplicações para .NET Compact Framework 1.0. Endereço: *www.microsoft.com/downloads/details.aspx?displaylang=pt-br&FamilyID=a5a02311-194b-4c00-b445-f92bec03032f*;
- *Windows CE 5.0 Device Emulator*: Emulador para testar as aplicações desenvolvidas com Compact Framework. Permite emular imagens de dispositivos com WindowsCE, Windows Mobile 2003 e 2003 SE para PocketPC e Smartphone. Endereço: *www.microsoft.com/downloads/details.aspx?familyid=A120E012-CA31-4BE9-A3BF-B9BF4F64CE72&displaylang=en*;
- *SDK for Windows Mobile 2003-based Pocket PCs*: Imagens de Windows Mobile 2003 para Pocket PC para o WindowsCE Emulator. Endereço: *www.microsoft.com/downloads/details.aspx?FamilyID=9996b314-0364-4623-9ede-0b5fbb133652&DisplayLang=en*;
- *Microsoft Device Emulator 2.0*: Emulador para Windows Mobile 5. Apesar de ser a versão 2.0, o instalador cria uma pasta 1.0. Endereço: *www.microsoft.com/downloads/details.aspx?FamilyID=dd567053-f231-4-a64-a648-fea5e7061303&DisplayLang=en*;
- *Microsoft Device Emulator Images*: Imagens de dispositivos para serem usadas com o Microsoft Device Emulator. Endereço: *www.microsoft.com/downloads/details.aspx?familyid=C62D54A5-183A-4A1E-A7E2-CC500ED1F19A&displaylang=en*;
- *Virtual Machine Network Driver for Microsoft Device Emulator*: Requerido para o Device Emulator, para permitir fazer uma rede entre o PC e o emulador. Endereço: *www.microsoft.com/downloads/details.aspx?familyid=dc8332d6-565f-4a57-be8c-1d4718d3af65&displaylang=en*;
- *Microsoft ActiveSync 4.5 BR*: Software utilizado para fazer sincronização de dados entre o PC e o dispositivo móvel. Com ele pode-se fazer transferência de dados também do PC para o emulador. Endereço: *www.microsoft.com/downloads/details.aspx?displaylang=pt-br&FamilyID=9e641c34-6f7f-404d-a04b-dc09f8141141*.
- *Compact Framework Build Helper 1.3 for BDS2006*: *Plugin* para o BDS2006 que facilita a construção de aplicações para Compact Framework enquanto o BDS não inclui um *Form Designer* específico para esse framework. Feche o BDS antes de iniciar a instalação. Endereço: *cc.codegear.com/item.aspx?id=24129* ou *www.jedsoftware.com/cf.htm*.

Baixando as ferramentas citadas anteriormente, não há obrigatoriedade de baixar as que mostrarei a seguir, pois já será possível testar as aplicações no WindowsCE Emulator ou Device Emulator, as ferramentas a seguir são apenas imagens adicionais de sistemas operacionais:

- *SDK for Windows Mobile 2003-based Smartphones*: Imagens de Windows Mobile 2003 para Smartphones para o WindowsCE Emulator. Endereço: *www.microsoft.com/downloads/details.aspx?familyid=a6c4f799-ec5c-427c-807c-4-c0f96765a81&displaylang=en*;
- *Windows Mobile 2003 Second Edition Emulator Images for Pocket PC*: Imagens de Windows Mobile 2003 SE para Pocket PC para o WindowsCE Emulator. Endereço: *www.microsoft.com/downloads/details.aspx?FamilyID=5c53e3b5-f2a2-47d7-a41d-825fd68ebb6c&DisplayLang=en*;

O *Windows Mobile 5.0 SDK for Pocket PC* requer o Visual Studio instalado, assim, não utilizaremos as imagens dele. Caso você possua o Visual Studio, você pode baixar o *Windows Mobile 5.0 SDK for Pocket PC* em *www.microsoft.com/downloads/details.aspx?familyid=83A52AF2-F524-4EC5-9155-717CBE5D25ED&displaylang=en*.

Utilizaremos o banco de dados PostgreSQL que pode ser baixado no site oficial em *www.postgresql.org*, mas como falei, você pode utilizar o banco que desejar. Para administrar o banco de dados pode-se utilizar a ferramenta *PgAdmin*, que vem junto com o instalador do banco de dados para Windows.

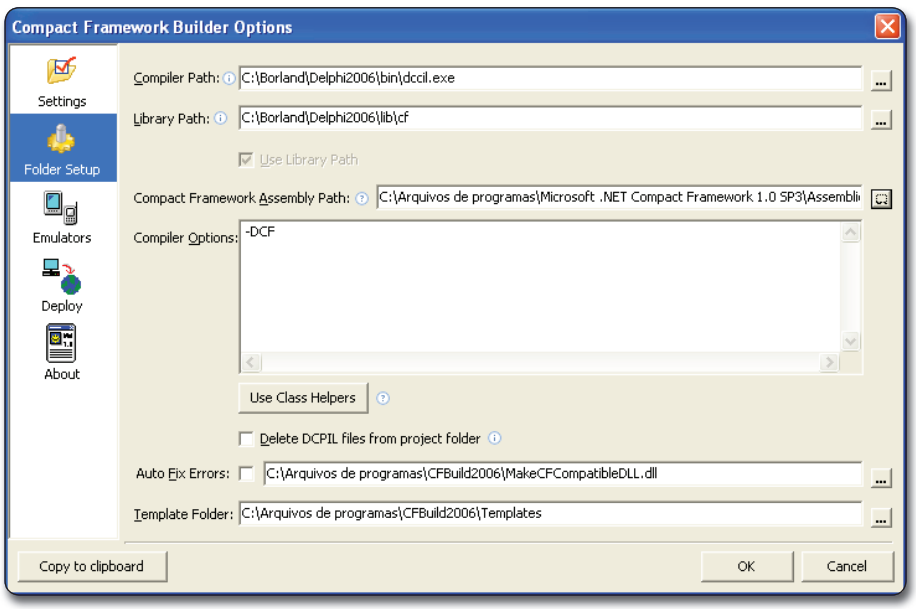


Figura 1. Configurando o CF Builder

Sugiro a ferramenta SQL Manager 2007 Lite for PostgreSQL, disponível para download em *www.sqlmanager.net/products/postgresql/manager*. A EMS, empresa que produz a ferramenta possui excelentes ferramentas de administração para diversos bancos de dados. Existe a versão paga e a versão lite, que é gratuita. A versão *Lite* não possui alguns recursos existentes na versão completa, mas mesmo assim, é uma excelente ferramenta. A instalação de todas as ferramentas são bastante simples, e dispensam comentários, basta usar as opções padrão.

Compact Framework Build Helper

Após instalar o *Compact Framework Build Helper* (a partir de agora o chamarei apenas de CF Builder), abra o BDS2006. Durante o carregamento será aberta uma janela para configuração do *plugin* como mostra a **Figura 1**.

Na **Tabela 1** temos a finalidade dos principais campos mostrados na figura.

Na pasta de instalação do CF Builder existem alguns documentos PDF e vídeo-aulas mostrando as configurações básicas do *plug-in*.

Incluindo um menu do CF Builder no Delphi

Vamos às outras configurações do CF Builder. Clique no botão *Settings* da janela

Campo	Finalidade
Compiler Path	Caminho do Compilador Delphi .Net
Library Path	Caminho onde estão os arquivos DCUIL para o Compact Framework. Esses arquivos são os antigos DCU só que compilados na *Microsoft Intermediate Language* (IL). Não é necessário informar.
Compiler Options	Define opções de compilação. Você pode clicar com o botão direito no campo para escolher as opções. O botão logo abaixo facilita o preenchimento desse campo. A opção "–DCF" cria um *Conditional Define* de nome CF, no momento de compilação do projeto, para identificar que o projeto será compilado para Compact Framework. Clicando com o botão direito no campo você pode selecionar também os *assemblies* do .NET Compact Framework que deseja utilizar. Como no BDS2006 não existem os arquivos DCPIL correspondentes aos *assemblies* do .NET CF, para serem utilizados na compilação do projeto, pois a Microsoft requer que você tenha o Visual Studio para compilar as aplicações para Compact Framework, precisaremos utilizar assemblies diferentes na compilação do projeto. Esses assemblies diferenciados do .NET CF estão compactados dentro da pasta de instalação do .NET CF 1.0 (normalmente em *C:\Arquivos de programas\Microsoft .NET Compact Framework 1.0 SP3*). Vamos utilizar os *assemblies* para processadores ARM, que são suportados pelos emuladores, descompactando o arquivo *netcf.core.ppc3.ARM.cab*. Precisamos renomear esses arquivos para os seus nomes reais. Você pode criar um arquivo BAT para executar os comandos a seguir e renomear os arquivos: ren mscorlib.011 mscorlib.dll ren 00system.012 System.dll ren SYD8C9~1.013 System.Drawing.dll ren SY17B8~1.014 System.Web.Services.dll ren SY5DD4~1.015 System.Windows.Forms.dll ren SY9B57~1.016 System.Windows.Forms.DataGrid.dll ren SY40C7~1.017 System.Xml.dll ren SYB769~1.018 System.Net.IrDA.dll ren SYSTEM~2.019 System.Data.dll ren MICROS~2.020 Microsoft.VisualBasic.dll ren MICROS~3.021 Microsoft.WindowsCE.Forms.dll del NETCF_~1.999 pause Você pode colocar os arquivos renomeados na pasta *C:\Arquivos de programas\Microsoft .NET Compact Framework 1.0 SP3\Assemblies* (você deve criar a mesma), por exemplo. No link para download deste artigo incluí o arquivo BAT com as linhas de comando anteriores.
Compact Framework Assembly Path	Caminho onde estão os assemblies do Compact Framework mostrados anteriormente, na pasta *C:\Arquivos de programas\Microsoft .NET Compact Framework 1.0 SP3\Assemblies*. É necessário informar o caminho dos assemblies pois no momento da compilação esses serão utilizados.
Use Class Helpers	Inclui no campo *Compiler Options* os principais *Class Helpers do CF Builder*. Como no .NET Framework existem muitas propriedades e métodos que não existem no .NET CF, e o Form Designer do BDS não suporta ainda o .NET CF, ele inclui no código essas propriedades e métodos que não são suportados pelo .NET CF para podermos criar o form usando o Form Designer atual. Os *Class Helpers* estendem as classes do .NET CF, sem criar novas classes, adicionando essas propriedades e métodos inexistentes no .NET CF, para permitir que o projeto seja compilado. A maioria dos atributos e métodos criados não terá nenhuma funcionalidade, servirá apenas para podermos usar o Form Designer, outros realmente são implementados para incluir alguns recursos.
Delete DCPIL files from project folder	Recriar os arquivos DCPIL a cada compilação. Isso é útil quando você estiver criando aplicações desktop e .NET CF a partir do mesmo código-fonte, o que não é o nosso caso.
Auto Fix Errors	Marcando essa opção o *MakeCFCompatibleDLL.dll* utilizará compilação condicional, usando a define CF mostrada anteriormente, para não compilar no código as propriedades, métodos e linhas do código-fonte que não são compatíveis com o .NET CF. Marque essa opção.

Tabela 1. Opções do CF Builder

do CF Builder e marque a opção *Install all Compact Framework Builder menu item into the one top level menu* (**Figura 2**), para que as opções do CF Builder sejam colocadas em um menu específico dentro do BDS.

Para a alteração ter efeito você precisa fechar e abrir novamente o BDS. Se essa opção não for marcada, os itens do CF Builder são colocados dentro dos menus *Project* e *Tools*.

Configurando os emuladores

Para configurar os emuladores clique no *Emulators* da janela do CF Builder. Clique em *New* para criar uma nova configuração de emulador. Na aba *General*, em *Name* informe um nome para o emulador.

Vamos utilizar uma imagem de WindowsCE utilizando o *WindowsCE Emulator*, assim, dê o nome de "WinCE Emu – WinCE OS". Você pode marcar a opção *Default* para indicar se esse será o emulador padrão para testar a aplicação.

No campo *Emulator Path* informe o caminho do WindowsCE emulator, que por padrão é *C:\Arquivos de programas\Windows CE 5.0 Emulator\ Emulator_500.exe*. No campo *OS Image* informe o caminho de uma imagem de sistema operacional para o emulador. Vamos utilizar uma imagem de WindowsCE disponível na pasta do emulador, por padrão é *C:\Arquivos de programas\Windows CE 5.0 Emulator\NK.cem*.

No campo *Shared Path* você pode indicar uma pasta no seu computador para ser compartilhada com o emulador. Essa pasta funcionará como um *Storage Card* (um cartão de memória) conectado ao dispositivo e assim você terá acesso, a partir do emulador, aos arquivos existentes na pasta no seu computador. Certifique-se de que a pasta tenha permissão de gravação para todos os usuários (**Figura 3**).

Clique na aba *Display*. No campo *Skin*, podemos escolher um *skin* para o emulador, ou você pode escolher uma resolução de tela definida para o mesmo. Clique no *Select* para escolher uma resolução padrão (**Figura 4**).

Clique na aba *Communication*. Para criar uma rede entre o seu computador e o emulador, escolha no campo *Ethernet*, a opção *Virtual Switch* para que o emulador use o primeiro dispositivo de rede encon-

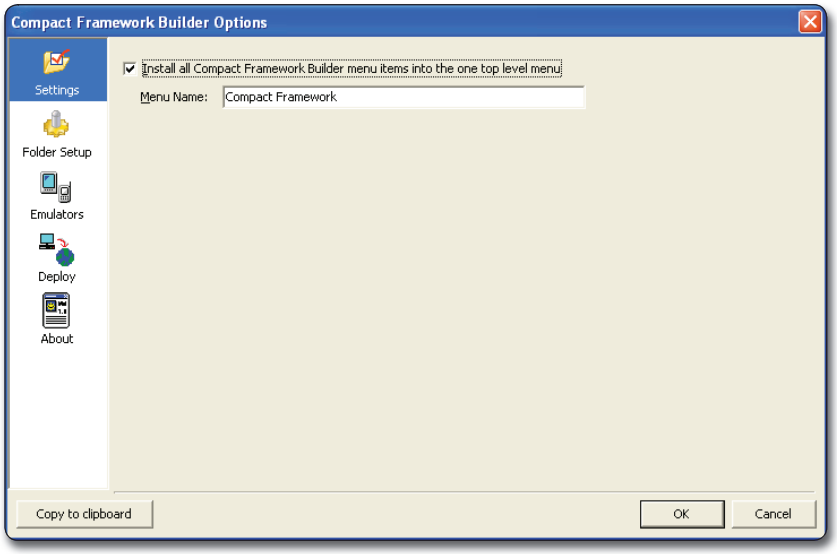


Figura 2. Menu do CF Builder no BDS

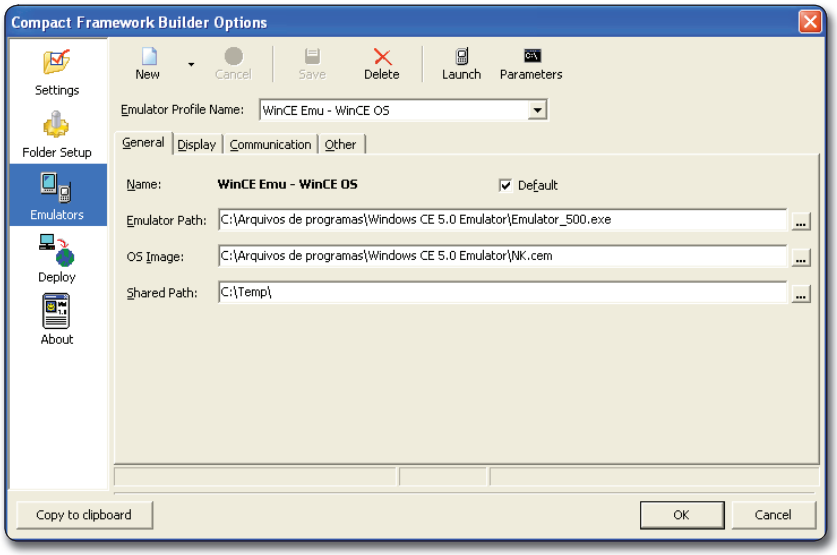


Figura 3. Configurando Emuladores

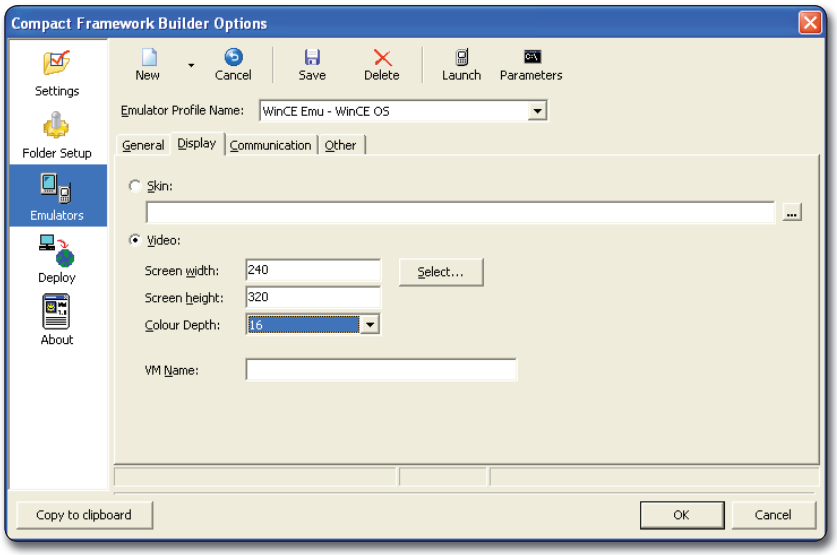


Figura 4. Aba Display do Emulador

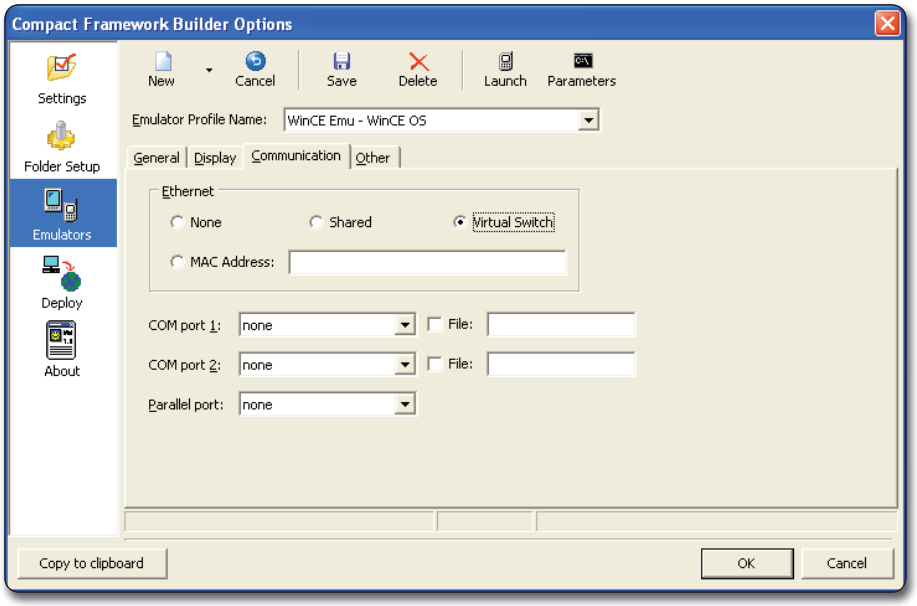


Figura 5. Aba Communication do Emulador

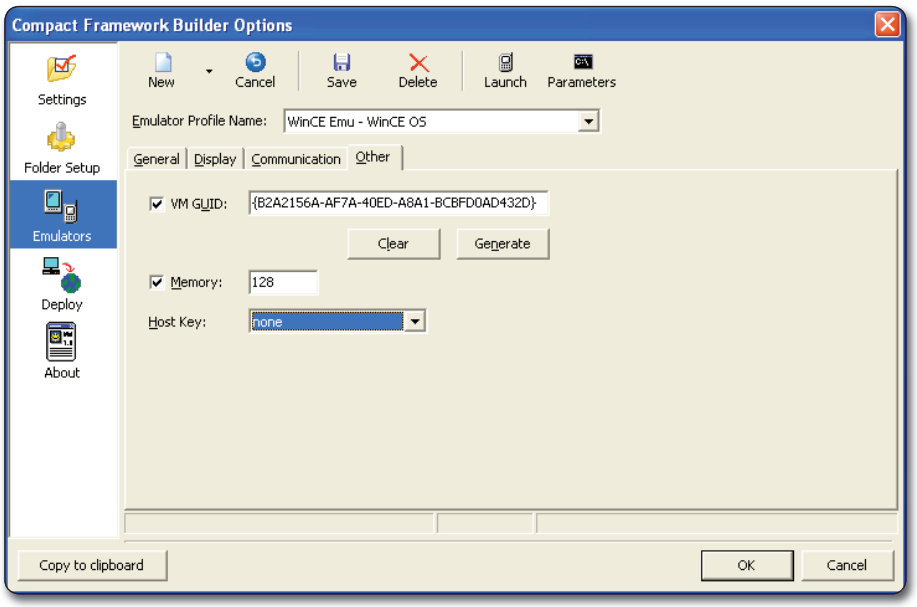


Figura 6. Aba Other do Emulador

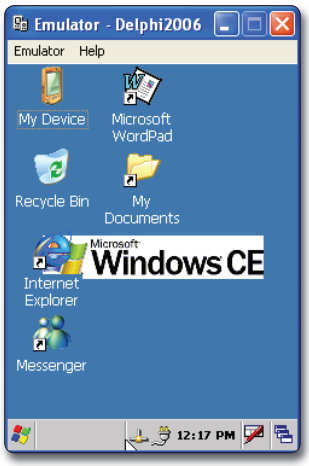


Figura 7. Emulador em execução

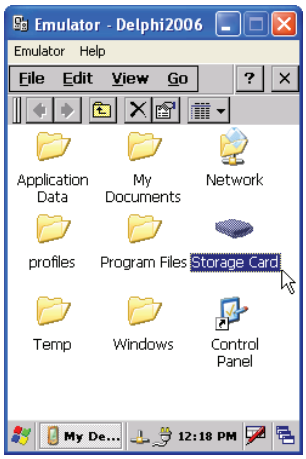


Figura 8. Storage Card no Emulador

trado no seu computador (**Figura 5**).

Clique na aba *Other*. Marque a opção VM GUID para que seja possível salvar o estado do sistema operacional rodando no emulador para que, na próxima vez que esse rodar, não tenha que dar *boot* novamente.

Clique em *Generate* para que seja gerado um identificador para ser associado à sessão do emulador. Assim, quando você rodar o emulador ele iniciará automaticamente no mesmo ponto em que parou na última execução. No campo *Memory* você pode especificar a quantidade de memória que deseja para dispositivo emulado (**Figura 6**).

Você pode clicar em *Parameters* para ver os parâmetros que são passados ao emulador, clique em *Save* para salvar as configurações. Clique em *Launch* para rodar o emulador (**Figura 7**).

A *Shared Folder* que indicamos na configuração do emulador pode ser acessada a partir do ícone *My Device* no desktop do emulador e depois no ícone *Storage Card*. Com isso você terá acesso à pasta no seu computador como se fosse um cartão de memória inserido no dispositivo móvel emulado (**Figura 8**).

Ao fechar o emulador você pode escolher *Turn Off* para desligar ou *Save state* para salvar o estado atual (caso tenha configurado a opção VM GUID no CF Builder, como mostrado anteriormente), assim, ao rodar o emulador novamente, ele iniciará rapidamente e no mesmo ponto onde foi finalizado na última execução.

Há um pequeno problema com a opção *Save state*, pois após rodar o emulador novamente, ele volta ao último estado, mas o *Storage Card* desaparece. Você pode utilizar o menu *Emulator>Soft Reset* da janela do emulador para reiniciar o dispositivo, fazendo o *Storage Card* aparecer novamente. Esse processo é mais rápido que a inicialização normal do emulador.

O CF Builder não está adequadamente preparado para utilizar o Device Emulator. Na janela de configuração dos emuladores são informados os parâmetros de linha de comando que devem ser passados ao emulador, porém, esses parâmetros são para o WindowsCE Emulator.

Se utilizarmos apenas as opções da aba

Aba	Campo	Valor
General	Name	"WinCE Emu – PocketPC 2003"
	Emulator Path	C:\Arquivos de programas\Windows CE 5.0 Emulator\Emulator_500.exe
	OS Image	C:\Arquivos de programas\Windows CE Tools\wce420\POCKET PC 2003\Emulation\PPC_2003_WWE.bin
	Shared Folder (coloque a pasta que desejar)	C:\Temp\Share
Display	Skin	C:\Arquivos de programas\Windows CE Tools\wce420\POCKET PC 2003\Emulation\Pocket_PC_Emulator_Skin.xml
Communication	Ethernet	Virtual Switch
Other	VM GUID	Marque esse campo e clique em *Generate* para gerar um identificador para gravar as sessões do sistema emulado, como feito na imagem de WindowsCE anteriormente.
Other	Memory	256

Tabela 2. Adicionando um novo item no CF Builder

General, que não são suficientes, conseguiremos rodar uma imagem de Windows Mobile 5.0 no Device Emulator, porém, não teremos como configurar outras opções utilizando o CF Builder. Assim, para usar o Microsoft Device Emulator, o executaremos a partir dos links criados no menu iniciar durante a instalação das imagens para Windows Mobile 5.0.

Vamos adicionar outro item para emular uma imagem de PocketPC 2003 com o WindowsCE Emulator. Assim, clique no *New* e preencha os campos como mostrado na **Tabela 2**.

Ao final, clique no botão *Save* para salvar as configurações e depois no *Launch* para rodar o emulador. Utilizando o WindowsCE Emulator, com qualquer imagem, ao reiniciar o emulador a partir de uma seção salva (se não desejar isso, desmarque a opção VM GUID na aba *Other*), o *Storage Card* desaparece (a pasta compartilhada – *Shared Folder*), assim, utilize o menu *Emulator>Soft Reset* do emulador para reiniciar o sistema e aparecer o *Storage Card*.

É melhor fazer assim, pois o *reset* demora menos do que ter que iniciar o sistema sem uma sessão salva e você não perde as configurações feitas. Para acessar o *Storage Card* em uma imagem de PocketPC, clique no menu *Start>Programs* dentro do emulador, abra o ícone *File Explorer*. Na parte inferior da janela do *File Explorer* existe um botão para acessar o *Storage Card*, como mostrado na **Figura 9**.

Configurando a rede no WindowsCE Emulator

Para configurar a rede entre seu computador e o emulador WindowsCE Emulator você precisa estar conectado à uma rede local ou à internet. O adaptador de rede do seu computador deve estar ativo. Se você estiver usando um notebook, por exemplo, onde você não tenha uma rede local nem conexão com a internet, você pode conectar o notebook a um *hub* (dispositivo de baixo custo) e assim conseguirá montar uma rede entre o notebook e o emulador.

Então, estando o ambiente físico pronto, clique no menu iniciar no emulador e escolha a opção *Settings>Network and Dial-up Connections*. Aparecerá um dispositivo de rede de nome DC211401. Clique com o botão direito nele e escolha *Properties*.

Devemos atribuir um IP ao dispositivo caso não estejamos conectados a uma rede com servidor DHCP para atribuir um IP automaticamente ao dispositivo. Você deve atribuir um IP que esteja na mesma faixa que o IP do seu computador, definir a máscara de rede (normalmente 255.255.255.0) e um *gateway* (caso tenha).

Veja exemplo de configuração na **Figura 10**, onde defini as configurações de IP do emulador, considerando que meu computador está conectado a um modem ADSL roteador com IP 10.1.1.1, meu PC possui IP 10.1.1.2, assim, atribuí o IP 10.1.1.3 para o emulador.

Clique no OK para confirmar as configurações. Após fechar o emulador, se você não salvar o estado atual, essas configurações serão perdidas. Quando o emulador faz um *boot* completo do sistema operacional do dispositivo móvel, o dispositivo de rede demora um pouco a funcionar. O ícone aparece na barra de tarefas do emulador.

Após algum tempo (demora um pouco) o ícone muda para indicando que o dispositivo de rede do emulador está funcionando. Para você conseguir navegar na internet a partir do emulador, você precisará configurar um endereço de servidor DNS na aba *Name Servers* da **Figura 10**.

Figura 9. WindowsCE Emulator rodando imagem de PocketPC 2003

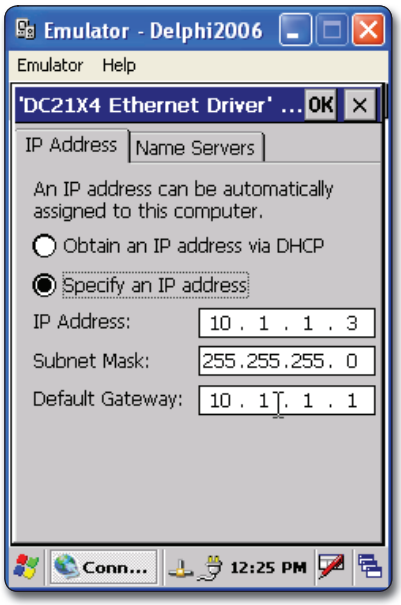


Figura 10. Configurações de rede do Emulador

Acessando uma página em um servidor Web

Após fazer as configurações de rede, o emulador poderá acessar páginas em servidores Web na rede em que ele foi configurado, ou na internet, se você tiver configurado o DNS. Você precisará instalar o *Internet Information Service* (IIS) no seu PC para construirmos nossos exemplos.

Criei um arquivo chamado *index.html* e coloquei na pasta do IIS, normalmente em *C:\inetpub\wwwroot*. A instalação, configuração, inicialização e utilização do IIS estão fora do escopo deste artigo.

Abra o Internet Explorer no emulador e digite o IP do seu computador para acessar o site hospedado no IIS. Veja a **Figura 11**, mostrando o emulador acessando o site no meu computador.

Se você estiver usando WindowsXP, será necessário liberar a porta 80 no *firewall* para conseguir acessar o servidor web. Para isso, abra o item *Firewall do Windows* no *Painel de Controle* do seu PC e, na aba *Exceções*, clique em *Adicionar Porta* e digite a porta 80 na lista de exceções, assim, o *firewall* não bloqueará essa porta.

Após o adaptador de rede estar habilitado no emulador, como mostrado anteriormente, pode levar ainda algum tempo para você conseguir acessar algum site a partir do mesmo.

Criando o banco de dados

Utilizaremos um banco de dados como mostrado no modelo entidade-relacionamento da **Figura 12**. Na **Tabela 3** temos a finalidade de cada tabela.

Crie um banco de dados na sua ferramenta de administração preferida do PostgreSQL e utilize o *script* da **Listagem 1** para criar as tabelas.

Tabela	Finalidade
pesquisa	Cadastrar as pesquisas a serem realizadas.
pergunta	Cadastrar as perguntas a serem feitas para cada pesquisa.
alternativa	Cadastrar as alternativas para cada pergunta.
entrevistado	Cadastrar dados sobre as pessoas entrevistadas.
resposta_entrevistado	Cadastrar as respostas das pessoas entrevistadas.

Tabela 3. Finalidade de cada tabela do banco

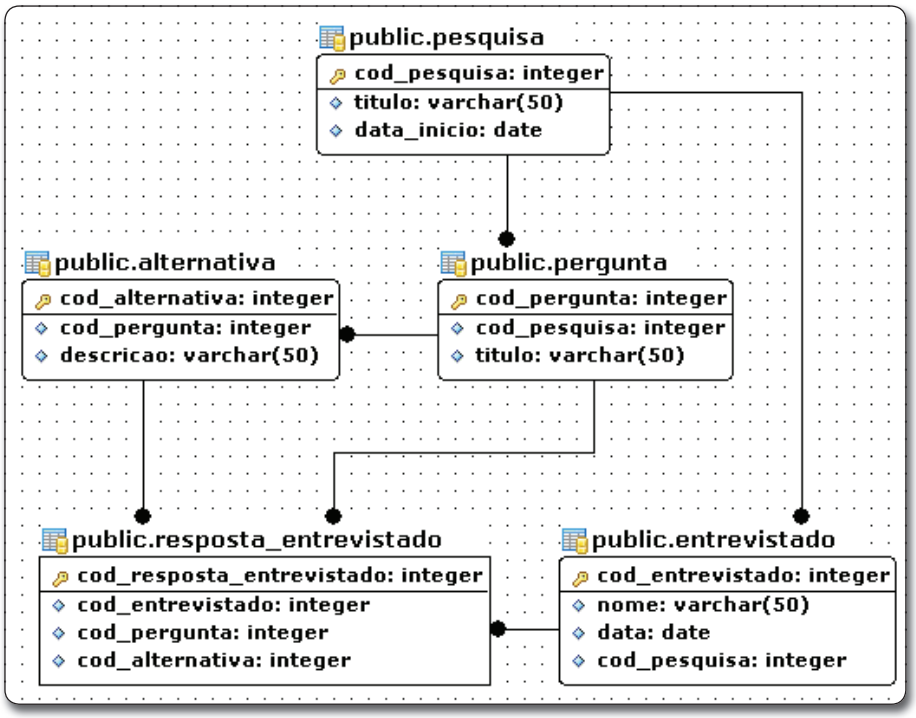


Figura 12. Modelo ER do Banco de Dados

Instalando os componentes para acesso ao PostgreSQL

Para acessar o PostgreSQL, utilizando ADO.NET, devemos adicionar os componentes no BDS. Estes componentes já são instalados quando o servidor PostgreSQL é instalado. Então, acesse o menu *Component>Installed .NET Components* e na janela que abre, no campo *Category* digite "PostgreSQL", para informar o nome da categoria onde os componentes serão adicionados no *Tool Palette*.

Em seguida clique em *Select an Assembly* e abra a pasta *Npgql*, dentro do diretório de instalação do PostgreSQL, normalmente em *C:\Arquivos de programas\PostgreSQL\8.0*. Na pasta *Npgsql* existe uma pasta *MS* e *Mono* indicando que os *assemblies* com os componentes existem tanto para o .NET Framework quanto para o Mono (uma versão do .NET Framework desenvolvida pela comunidade de software livre, que permite rodar aplicações .NET em outras plataformas, como Linux, por exemplo).

Entre na pasta *MS* e selecione o *Npgsql.dll*. Depois clique em OK. Veja os componentes instalados na **Figura 13**.

Criando o Web Service para acesso ao banco

A aplicação fará acesso ao banco de dados por meio de um Web Service. As-

Figura 11. Emulador acessando site no meu computador

sim, os componentes *NpgSQL* não serão utilizados na aplicação desktop, pois a camada de acesso a dados fica somente no Web Service.

Cria um projeto *ASP.NET Web Service Application* utilizando o menu *File>New>Other*. Na aba *Delphi for .NET Projects* escolha o item *ASP.NET Web Service Application* (**Figura 14**).

Na próxima tela que é aberta você deve informar o nome e caminho da aplicação. Em *Name* digite "PesquisaOpiniaoWS" e em *Location* escolha uma pasta onde salvar o projeto.

Para não ter problemas de permissão, escolha uma pasta dentro da pasta do servidor IIS (normalmente em *C:\inetpub\wwwroot*). Clique em OK para criar o projeto.

Clique no arquivo *WebService1.asmx* no *Project Manager* (canto superior direito), pressione F2 e altere seu nome para "PesquisaOpiniao.asmx". Clique no Form Designer para alterar o nome da classe do WebService de *TWebService1* para "TPesquisaOpiniaoWS".

Adicione um *NpgsqlConnection*. Altere sua propriedade *Name* para "Conn" e a propriedade *ConnectionString* para "SERVER=127.0.0.1; DATABASE=pesquisa_opiniao_clube_delphi; USER ID=postgres; PASSWORD=postgres; POLLING=True; MINPOOLSIZE=10; MAXPOOLSIZE=100;".

Listagem 1. Script SQL para criação do banco de dados

```
/* Os campos do tipo serial são campos inteiros que funcionam como campos auto-incremento
por meio de um sequence (como um generator do Firebird). Quando definimos um campo do tipo
serial, um sequence para o campo é criado automaticamente. */

create table pesquisa (
  cod_pesquisa serial not null,
  titulo varchar(50) not null,
  data_inicio  date not null,
  constraint pk_pesquisa
  primary key(cod_pesquisa)
);

create table pergunta (
  cod_pergunta serial not null,
  cod_pesquisa integer not null,
  titulo varchar(50) not null,
  constraint pk_pergunta primary key (cod_pergunta),
  constraint fk_pergunta_pesquisa
    foreign key (cod_pesquisa)
    references pesquisa(cod_pesquisa)
);

create table alternativa (
  cod_alternativa serial not null,
  cod_pergunta integer not null,
  descricao varchar(50) not null,
  constraint pk_alternativa
    primary key (cod_alternativa),
  constraint fk_alternativa_pergunta
    foreign key (cod_pergunta)
    references pergunta(cod_pergunta)
);

create table entrevistado (
  cod_entrevistado serial not null, nome varchar(50) not null,
  data date not null default current_date, cod_pesquisa integer not null,
  constraint pk_entrevistado primary key (cod_entrevistado),
  constraint fk_entrevistado_pesquisa foreign key (cod_pesquisa)
     references pesquisa(cod_pesquisa)
);

create table resposta_entrevistado (
  cod_resposta_entrevistado serial not null, cod_entrevistado integer not null,
  cod_pergunta integer not null, cod_alternativa integer not null,
  constraint pk_resposta_entrevistado primary key(cod_resposta_entrevistado),
  constraint fk_resposta_entrevistado_entrevistado
    foreign key (cod_entrevistado)
    references entrevistado(cod_entrevistado) on delete cascade,
  constraint fk_resposta_entrevistado_pergunta
    foreign key (cod_pergunta)
    references pergunta(cod_pergunta),
  constraint fk_resposta_entrevistado_alternativa
    foreign key (cod_alternativa)
    references alternativa(cod_alternativa));
```

Figura 13. Componentes Npgsql instalados

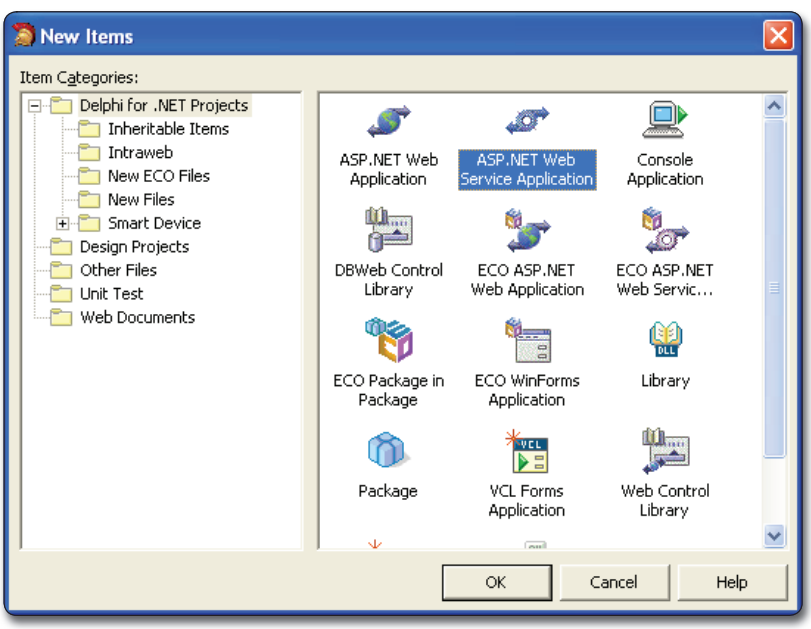


Figura 14. Criando um projeto ASP.NET WebService Application

Listagem 2. Função GetPesquisa

```
var
  cmd: NpgsqlCommand;
  da: NpgsqlDataAdapter;
begin
  { Abre a conexão com o banco }
  conn.Open;
  try

    { Instancia o DataSet que será retornado pela função }
    Result := DataSet.Create;
    cmd := NpgsqlCommand.Create('select * from '+
      'pesquisa where titulo like :titulo limit 1', conn);

    { Adiciona o parâmetro com o seu valor no cmd.
      Observe que foi utilizado o operador % para permitir busca parcial }
    cmd.Parameters.Add('titulo', Titulo + '%');

    { Instancia um NpgsqlDataAdapter que é capaz de gerar um DataSet a partir do
      resultado de um select }
    da := NpgsqlDataAdapter.Create(cmd);

    { Preenche o DataSet com os dados retornados pelo select no cmd }
    da.Fill(Result);
  finally
    { Fecha a conexão com o banco de dados }
    conn.Close;
  end;
end;
```

Figura 15. Armazenando a conexão do banco no web.config

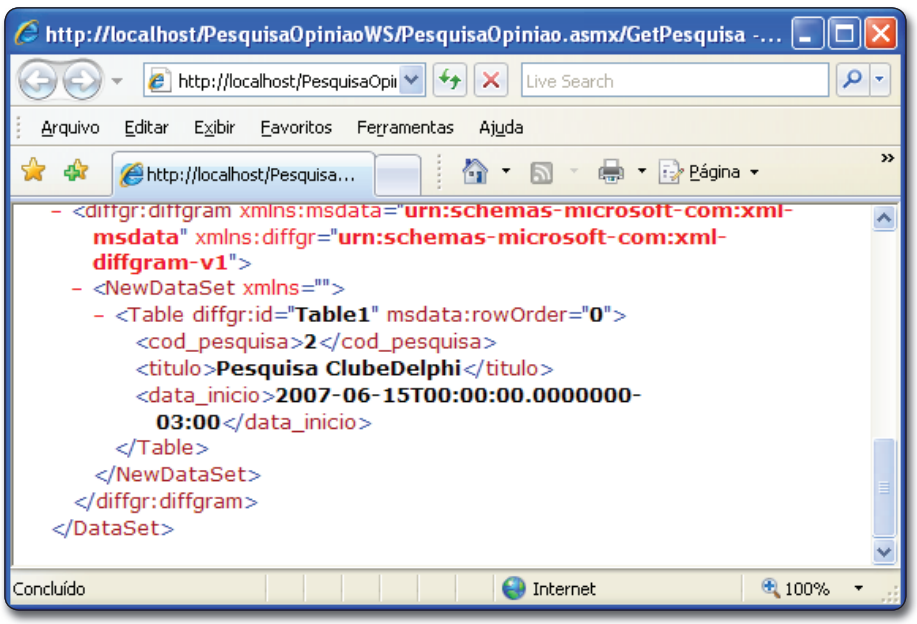


Figura 16. XML retornado pela função GetPesquisa

Considerando que seu servidor é local (127.0.0.1), o nome de usuário é *postgres* (o padrão), a senha é *postgres* e o banco de dados é *pesquisa_opiniao_clube_delphi*. Altere para as suas configurações.

Você pode clicar no botão ao lado da propriedade para abrir a janela de edição, porém, nessa janela não existem opções para configurar o Pool de Conexões (POLLING=True habilita o pool de conexões, MINPOOLSIZE=10 define a quantidade mínima de conexões que serão pré-estabelecidadas, MAXPOOLSIZE=100 define a quantidade máxima de conexões permitidas). Na pasta de instalação do *Npgsql*, mostrada anteriormente, existem arquivos HTML de ajuda.

Abra a propriedade *DynamicProperties* do *NpgsqlConnection* e clique no botão ao lado da sub-propriedade *ConnectionString*.

Na janela que é aberta, marque a opção *Map property to a key in configuration file* para que a configuração da conexão seja gravada no arquivo *web.config* e seja possível alterá-la sem recompilar a aplicação. O campo *Key* será o nome do identificador criado dentro do arquivo *web.config* (**Figura 15**).

Criando as funções no WebService

Para criar uma função no WebService é bastante simples. Vamos iniciar criando uma função para retornar os dados das pesquisas de opinião que serão utilizadas no dispositivo móvel.

Assim, criaremos uma função que receberá o título da pesquisa e retornará um *DataSet* contendo os dados retornados por um *select* no banco de dados. Para que as funções criadas sejam acessíveis externamente, é necessário incluir o atributo *[WebMethod]* acima do cabeçalho das funções. Assim, na seção *public* da classe do WebService, insira o código a seguir:

```
[WebMethod]
function GetPesquisa(Titulo: string):
DataSet;
```

Pressione CTRL+SHIFT+C para criar o corpo da função. Veja o código dela na **Listagem 2**.

A função instancia um *NpgsqlCommand* para executar instruções SQL. Note que foi criado um parâmetro dentro da instrução SQL para que sejam retornados apenas a pesquisa que coincidir com o título passado por parâmetro. A ins-

trução *limit 1* do PostgreSQL fará com que seja retornado apenas o primeiro registro, ou seja, somente uma pesquisa será retornada.

Para testar a função você pode rodar a aplicação. É aberto o navegador mostrando as funções existentes no Web Service. Clique em *GetPesquisa*. Como deve ser passado um parâmetro a ela, é mostrada outra página com um campo para cada parâmetro solicitado pela função.

Cadastre algumas pesquisas no banco e depois, informe um valor para o parâmetro solicitado e clique no *Invoke*. É aberta outra página exibindo o código XML do *DataSet* retornado pela função, contendo os dados das pesquisas retornadas (**Figura 16**).

Vamos implementar todas as outras funções que necessitaremos no Web Service. Na **Listagem 3**, temos a assinatura das funções.

Note que existe, no atributo *WebMethod*, uma propriedade *Description* que descreve a finalidade da função, expondo esse texto para os desenvolvedores que criarão as aplicações para consumir o Web Service, criando uma documentação para o mesmo.

Pressione CRTL+SHIFT+C para criar o corpo das funções e rode o Web Service para ver como a página ficará. Na **Listagem 4** temos o código das funções que devem ser criadas. O código está todo comentado para melhor entendimento.

As funções desse Web Service podem ser acessadas por qualquer um que conheça o endereço do mesmo, porém, no caso da nossa aplicação, isso não é algo desejado. O Web Service deve ter alguma forma de autenticação para permitir a execução das funções.

A forma mais fácil de fazer isso é incluindo em todas as funções um parâmetro login e senha para que a função faça a autenticação, executando ou não o seu código. Pode-se criar uma função privada, sem o atributo *[WebMethod]* para ser chamada a partir das outras funções do Web Service e verificar a autenticação do usuário.

Não implementarei isso para não estender o artigo e por ser algo bem simples de fazer. Qualquer alteração no código do Web Service, esse deve ser recompilado. Se forem modificados parâmetros das funções ou adicionadas ou removidas novas funções, você deve clicar com o botão direito na *PesquisaOpiniaoServer* da aplicação cliente, no *Project Manager*, e escolher a opção *Update Web Reference* para atualizar as referências às funções do WebService automaticamente.

Listagem 3. Assinatura das novas funções do WebService

```
[WebMethod (Description=
  'Recupera o valor de um sequence no banco de dados, para ser utilizado nas tabelas '+
  'que possuem cardinalidade 1..N, como o caso de entrevistado e resposta_entrevistado ')]
function GetPrimaryKeyNextVal(TableName: string): Integer;

[WebMethod (Description=
  'Retorna os dados do entrevistado cujo nome é passado por parâmetro. Poderia-se ' +
  'utilizar outros parâmetros para a função, pois podem ter várias pessoas com o mesmo nome ' +
  'mas, para simplificar, coloquei o Nome ')]
function GetEntrevistado(Nome: string): DataSet;

[WebMethod (Description=
  'Inclui os dados de um entrevistado no banco. Retorna o código do entrevistado incluído, ' +
  'usando a função GetPrimaryKeyNextVal ')]
function IncluirEntrevistado(Nome: string;
  CodPesquisa: Integer): Integer;

[WebMethod (Description=
  'Retorna os dados de uma resposta do entrevistado. Essa função é usada primariamente ' +
  'para retornar um DataSet que contém a estrutura da tabela resposta_entrevistado para a aplicação '+
  'cliente. Esse DataSet é usado para permitir o cadastro de respostas ')]
function GetRespostaEntrevistado(
  CodRespostaEntrevistado: Integer): DataSet;

[WebMethod (Description=
  'Retorna as perguntas cadastradas para uma pesquisa ')]
function GetPerguntas(CodPesquisa: Integer): DataSet;

[WebMethod (Description=
  'Retorna todas as alternativas de todas as perguntas de uma pesquisa.')]
function GetAlternativas(CodPesquisa: Integer): DataSet;

[WebMethod (Description=
  'Registra uma resposta de um entrevistado. A função verifica se a pergunta passada por ' +
  'parâmetro já foi respondida ou não. Se já foi, atualiza a resposta, se não, inclui' )]
function RegistrarResposta(CodEntrevistado,
  CodPergunta, CodAlternativa: Integer): Boolean;
```

Conclusão

Nesta primeira parte do artigo vimos como baixar os softwares necessários para o desenvolvimento de aplicativos móveis com o Delphi, como trabalhar com o Compact Framework Builder, criar o banco de dados e definir um Web Service para manipular os dados. Na próxima edição veremos como construir a aplicação no PocketPC que fará acesso a esse servidor. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma série de vídeo aulas de Jefferson Junglaus, sobre o Compact Framework no Delphi.

www.devmedia.com.br/articles/listcomp.asp?txtsearch=Compact+Framework+no+Delphi

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

Listagem 4. Funções adicionais a serem implementadas no WebService

```
uses System.Text;
...
function TPesquisaOpiniaoWS.IncluirEntrevistado(
  Nome: string; CodPesquisa: Integer): Integer;
var
  cmd: NpgsqlCommand;
  CodEntrevistado: Integer;
begin
  conn.Open;
  try
    { Pega o valor para a chave primária da tabela }
    CodEntrevistado := GetPrimaryKeyNextVal('entrevistado');
    Cmd := NpgsqlCommand.Create(' insert into '+
      'entrevistado (cod_entrevistado, nome, ' +
      'cod_pesquisa) values (:cod_entrevistado, ' +
      ':nome, :cod_pesquisa) ', conn);
    cmd.Parameters.Add('cod_entrevistado', CodEntrevistado.ToString);
    cmd.Parameters.Add('nome', nome);
    cmd.Parameters.Add('cod_pesquisa', CodPesquisa.ToString);
    { Se executou com sucesso, retorna o código gerado }
    if cmd.ExecuteNonQuery > 0 then
      Result := CodEntrevistado
    else Result := 0;
  finally
    conn.Close;
  end;
end;

function TPesquisaOpiniaoWS.RegistrarResposta(
  CodEntrevistado, CodPergunta, CodAlternativa: Integer): Boolean;
var
  sql: string;
  CodRespostaEntrevistado: Integer;
  cmd: NpgsqlCommand;
begin
  { Para não aumentar a quantidade de código e estendendo muito o
    artigo, não foi utilizada a classe StringBuilder, que faz melhor
    uso de memória na concatenação de strings }
  { Faz um select para verificar se o registro já existe }
  sql := 'select cod_resposta_entrevistado from '+
    'resposta_entrevistado where cod_entrevistado = :cod_entrevistado '+
    'and cod_pergunta = :cod_pergunta ';
  conn.Open;
  try
    cmd:= NpgsqlCommand.Create(sql.ToString, conn);
    cmd.Parameters.Add('cod_entrevistado', CodEntrevistado.ToString);
    cmd.Parameters.Add('cod_pergunta', CodPergunta.ToString);
    CodRespostaEntrevistado := Convert.ToInt32(cmd.ExecuteScalar);
    cmd.Parameters.Clear;
    { Se o registro já existe, atualiza }
    if CodRespostaEntrevistado > 0 then
    begin
      cmd.CommandText := 'update '+
        'resposta_entrevistado set cod_alternativa = '+
        ':cod_alternativa where cod_resposta_entrevistado = '+
        ':cod_resposta_entrevistado';
       cmd.Parameters.Add('cod_alternativa', CodAlternativa.ToString);
       cmd.Parameters.Add('cod_resposta_entrevistado',
         CodRespostaEntrevistado.ToString);
    end
    { Se o registro não existe, insere }
    else
    begin
      cmd.CommandText := 'insert into '+
        'resposta_entrevistado (cod_entrevistado, '+
        'cod_pergunta, cod_alternativa) ' +
        'values (:cod_entrevistado, :cod_pergunta, :cod_alternativa)';
      cmd.Parameters.Add('cod_entrevistado', CodEntrevistado.ToString);
      cmd.Parameters.Add('cod_pergunta', CodPergunta.ToString);
      cmd.Parameters.Add('cod_alternativa', CodAlternativa.ToString);
    end;
    Result := cmd.ExecuteNonQuery > 0;
  finally
    conn.close;
  end;
end;

function TPesquisaOpiniaoWS.GetAlternativas(CodPesquisa: Integer): DataSet;
var
  cmd: NpgsqlCommand;
  da: NpgsqlDataAdapter;
begin
  Result := DataSet.Create;
  conn.Open;
  try
    cmd:= NpgsqlCommand.Create(
      'select a.* from alternativa a ' +
      ' inner join pergunta p on p.cod_pergunta = a.cod_pergunta ' +
      ' where p.cod_pesquisa = :cod_pesquisa', conn);
    cmd.Parameters.Add('cod_pesquisa', CodPesquisa.ToString);
    Da := NpgsqlDataAdapter.Create(cmd);
    da.Fill(result);
  finally
    conn.Close;
  end;
end;

function TPesquisaOpiniaoWS.GetEntrevistado(Nome: string): DataSet;
var
  cmd: NpgsqlCommand;
  da: NpgsqlDataAdapter;
begin
  conn.Open;
  try
    Result := DataSet.Create;
    cmd:= NpgsqlCommand.Create(
      'select * from entrevistado where nome like ' +
      ':nome order by cod_entrevistado desc limit 1', conn);
    cmd.Parameters.Add('nome', nome + '%');
    da := NpgsqlDataAdapter.Create(cmd);
    da.Fill(result);
  finally
    conn.close;
  end;
end;

function TPesquisaOpiniaoWS.GetPerguntas(CodPesquisa: Integer): DataSet;
var
  cmd: NpgsqlCommand;
  da: NpgsqlDataAdapter;
begin
  Result := DataSet.Create;
  conn.Open;
  try
    cmd:= NpgsqlCommand.Create(
      'select * from pergunta where cod_pesquisa = '+
      ':cod_pesquisa', conn);
    cmd.Parameters.Add('cod_pesquisa', CodPesquisa.ToString);
    da := NpgsqlDataAdapter.Create(cmd);
    da.Fill(result);
  finally
    conn.Close;
  end;
end;

function TPesquisaOpiniaoWS.GetPrimaryKeyNextVal(
  TableName: string): Integer;
var
  cmd: NpgsqlCommand;
  sql: StringBuilder;
  fechar: boolean;
begin
  fechar := false;
  sql:= StringBuilder.Create('select nextval(''');
  sql.Append(TableName);
  sql.Append('_cod_');
  sql.Append(TableName);
  sql.Append('_seq'')');

  { Se a conexão está fechada, abre e indica
    que ao final é pra fechar (fechar := true;) }
  if conn.State = ConnectionState.Closed then
  begin
    conn.open;
    fechar := true;
  end;
  try
    cmd:= NpgsqlCommand.Create(sql.ToString, conn);
    result:= Convert.ToInt32(cmd.ExecuteScalar);
  finally
    if fechar then
      conn.close;
  end;
end;

function TPesquisaOpiniaoWS.GetRespostaEntrevistado(
  CodRespostaEntrevistado: Integer): DataSet;
var
  cmd: NpgsqlCommand;
  da: NpgsqlDataAdapter;
begin
  conn.Open;
  try
    Result := DataSet.Create;
    cmd := NpgsqlCommand.Create(
      'select * from resposta_entrevistado  ' +
      'where cod_resposta_entrevistado = '+
      ':cod_resposta_entrevistado', conn);
    cmd.Parameters.Add('cod_resposta_entrevistado',
      CodRespostaEntrevistado.ToString);
    da := NpgsqlDataAdapter.Create(cmd);
    da.Fill(result);
  finally
    conn.close;
  end;
end;
```

NOVOS PLANOS

Descubra porque Hospedix é a hospedagem preferida dos profissionais.

Plano **Profissional** 1

1 Registro de domínio grátis*
3 Domínios por conta
50 GB de transferência
30 GB para e-mails com SSL
1 GB de espaço
ASP, ASP.NET 2.0 e PHP 5
Access e MySQL 5
SQL Server 2005 Express
Painel de controle e webmail
Editor de páginas on-line
Anti-spam e Anti-vírus

R$ 25,90 por mês

30 DIAS GRÁTIS
PARA EXPERIMENTAR
Ao assinar, digite o código de desconto abaixo exclusivo para leitores da .NET Magazine e ganhe o primeiro mês inteiramente grátis.

RVSTNET

DOMÍNIO GRÁTIS
ENQUANTO FOR CLIENTE
Só na Hospedix você tem registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

PAINEL DE CONTROLE
PLESK 8.1 PROFISSIONAL
Painel de controle completo onde você mesmo adiciona domínios, sub-domínios, cria contas de e-mail, e muito mais.

Transfira seu site hoje mesmo para a Hospedix e descubra porque somos a empresa de hospedagem preferida dos profissionais.

www.hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

As promoções anunciadas não são cumulativas com outras promoções | * Domínio .com .net .org ou .info grátis e isento de taxas anuais enquanto for cliente Hospedix.

Artigos intermediários sobre Delphi Win32 e Delphi .NET

SEÇÃO

Contas a Pagar e Cobrança

Crie um sistema completo com Delphi, Firebird 2.0 e dbExpress - Parte 3

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Nessa terceira parte do nosso mini-curso do Sistema de Contas a Pagar e Cobrança, onde estamos desenvolvendo o aplicativo *SysPague*, veremos como criar a tela de cadastro de Contas a Receber. Veremos também como criar um recurso bastante útil ao usuário: duplicação de lançamentos de contas a pagar e a receber.

Esse recurso pode ser utilizado para duplicar contas que se repetem naturalmente todo mês (contas parceladas), tais como financiamentos, *leasings*, compras a prazo e parcelamentos de modo geral. Esse recurso foi recomendado pelo leitor Francisco J. Riva que enviou e-mail com a dica.

Daremos continuidade ao curso desenvolvendo o cadastro de contas a receber como mencionado anteriormente, para isso precisamos alterar o banco de dados incluindo mais uma tabela.

Alterando o banco de dados de exemplo

Assim como fizemos nos artigos anteriores criaremos mais uma tabela no banco de dados utilizando o aplicativo IBExpert. Após conectar ao banco de dados, clique com o botão direito sobre o item *Tables* e escolha a opção *New Table*. No alto da janela, aparecerá o nome da tabela sugerido, troque-o para "Contas_Receber".

Em seguida para criar um campo, digite o nome em *Field Name*, selecione seu tipo em *Field Type* e marque o item *Not Null* (caso seja necessário). Caso prefira, execute o *script* da **Listagem 1**.

Note que estamos criando uma tabela exatamente igual a criada no artigo anterior (*Contas_Pagar*). Poderíamos ter criado uma única tabela onde diferenciaríamos cada registro usando uma *flag* para indicar se o registro é uma entrada ou saída, ou seja, um campo que recebesse tal indicação.

Resolvi criar tabelas diferentes, assim temos tranqüilidade no futuro, caso ocorram mudanças de conceito em nossa aplicação que façam com que ambos os lançamentos se diferenciem demais um do outro. Também estamos prevendo a criação das chaves primárias e chaves estrangeiras, necessárias para garantir a integridade do sistema.

Precisamos apenas entender que cada lançamento em contas a receber é referente a cada fatura disparada contra um cliente, ou seja, é entrada de dinheiro em nosso fluxo de caixa. Isso significa que nossas chaves estrangeiras precisam ser vinculadas a tabela CLIENTES.

No código a seguir (extraído da **Listagem 1**), criamos uma chave estrangeira incluindo BANCO, AGENCIA e CONTA em que a fatura foi paga. Isso é importante para controlarmos o saldo bancário da conta:

```
ALTER TABLE CONTAS_RECEBER ADD CONSTRAINT
  FK_CONTAS_RECEBER_BANCO FOREIGN KEY
  (BANCO, AGENCIA, CONTA) REFERENCES CONTAS
  (BANCO, AGENCIA, CONTA) ON UPDATE CASCADE;
```

Em seguida estamos criando uma nova chave estrangeira, dessa vez com o campo CNPJ vinculado-o à tabela CLIENTES. Assim toda vez que alterarmos o CNPJ de um cliente, os registros em contas a receber, vinculados ao cliente, serão automaticamente alterados:

```
ALTER TABLE CONTAS_RECEBER ADD CONSTRAINT
  FK_CONTAS_RECEBER_CNPJ FOREIGN KEY (CNPJ)
  REFERENCES CLIENTES (CNPJ) ON UPDATE
CASCADE;
```

Alterando o projeto

Vamos passar rapidamente pelos passos de alteração do projeto e preparação para inclusão da nova tela de cadastro do sistema. O primeiro passo é alterar o *dmdPrincipal* assim como fizemos no artigo anterior.

No Data Module devemos incluir três novos componentes de acesso a dados. Um *SQLQuery* ("sqlContas_Receber"), um *DataSetProvider* ("dspContas_Receber") e um *ClientDataSet* ("cdsContas_Receber"). A configuração de cada um segue os mesmos padrões dos artigos anteriores, bastando apenas modificar a instrução SQL do *sqlContas_Receber* e modificar o nome dos componentes. A instrução SQL para a tabela *Contas_Receber* é bastante semelhante a de *Contas_Pagar*, como podemos ver na **Listagem 2**.

Listagem 1. Criação da tabela e índices da Contas_Receber

```
CREATE TABLE CONTAS_RECEBER (
  CODIGO        VARCHAR(20) NOT NULL,
  CNPJ          VARCHAR(18) NOT NULL,
  DESCRICAO     VARCHAR(60) NOT NULL,
  VLR_REAL      NUMERIC(15,2),
  VLR_PAGO      NUMERIC(15,2),
  JUROS         NUMERIC(15,2),
  MORA          NUMERIC(15,2),
  DT_CADASTRO   TIMESTAMP,
  DT_VECTO      TIMESTAMP,
  DT_PAGTO      TIMESTAMP,
  BANCO         INTEGER,
  AGENCIA       VARCHAR(10),
  CONTA         VARCHAR(10),
  STATUS        VARCHAR(1),
  DT_ALTERACAO  TIMESTAMP
);

ALTER TABLE CONTAS_RECEBER ADD CONSTRAINT PK_CONTAS_RECEBER PRIMARY KEY (CODIGO, CNPJ);
ALTER TABLE CONTAS_RECEBER ADD CONSTRAINT FK_CONTAS_RECEBER_BANCO FOREIGN KEY
  (BANCO, AGENCIA, CONTA) REFERENCES CONTAS
  (BANCO, AGENCIA, CONTA) ON UPDATE CASCADE;
ALTER TABLE CONTAS_RECEBER ADD CONSTRAINT FK_CONTAS_RECEBER_CNPJ FOREIGN KEY (CNPJ)
  REFERENCES CLIENTES (CNPJ) ON UPDATE CASCADE;
```

Listagem 2. Instrução SQL do sqlContas_Receber

```
SELECT
  CODIGO, CNPJ, DESCRICAO, VLR_REAL, VLR_PAGO, JUROS, MORA, DT_CADASTRO, DT_VECTO,
  DT_PAGTO, BANCO, AGENCIA, CONTA, STATUS, DT_ALTERACAO
FROM CONTAS_RECEBER
WHERE STATUS <> 'I'
ORDER BY DESCRICAO
```

Listagem 3. Função PesquisarContasReceber

```
function TdmdPrincipal.PesquisarContasReceber(ACampo, AValor: string;
  ATipoPesquisa: Integer): Boolean;

var
  Operador: string;
  strAux: string;

begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlContas_Receber, SQL do
  begin
    cdsContas_Receber.Close;
    Clear;
    Params.Clear;
    Add('SELECT');
    Add('  CODIGO, CNPJ, DESCRICAO, VLR_REAL, VLR_PAGO, JUROS, MORA, DT_CADASTRO,');
    Add('  DT_VECTO, DT_PAGTO, BANCO, AGENCIA, CONTA, STATUS, DT_ALTERACAO');
    Add('FROM');
    Add('  CONTAS_RECEBER');
    Add('WHERE ');

    { Monta a cláusula where }
    Add('(' + ACampo + ' ' + Operador + ' :' + ACampo + ')');
    cdsContas_Receber.FetchParams;

    { A variável strAux foi criada para tratar o valor que está chegando no parâmetro
      AValor. Ele não pode ser maior (em caracteres) do que o tamanho do campo, pois neste
      caso é gerado uma exceção. }

    case ATipoPesquisa of
      0: cdsContas_Receber.Params.ParamByName(ACampo).AsString := AValor;
      1:
        begin
          strAux := Copy(AValor, 1, cdsContas_Receber.FieldByName(ACampo).Size - 1);
          cdsContas_Receber.Params.ParamByName(ACampo).AsString := strAux + '%';
        end;
      2:
        begin
          strAux := Copy(AValor, 1,
            cdsContas_Receber.FieldByName(ACampo).Size - 2);
          cdsContas_Receber.Params.ParamByName(ACampo).AsString := '%' + strAux + '%';
        end;
    end;

    Add('  AND STATUS <> ''I'' ');
    Add('ORDER BY');
    Add('  CODIGO');
    cdsContas_Receber.Open;
    Result := not cdsContas_Receber.IsEmpty;
  end;
end;
```

A segunda providência importante, é fazer a criação da função usada no *box* de pesquisa da janela Cadastro de Contas a Receber que criaremos mais adiante. Use o código da **Listagem 3** para criar a *PesquisarContasReceber* no Data Module.

O terceiro e último passo referente às alterações no Data Module, diz respeito aos eventos e adição dos campos da tabela *Contas_Receber* nos componentes *sqlContas_Receber* e *cdsContas_Receber*.

Após incluir a instrução SQL no *sqlContas_Receber* clique duas vezes no mesmo e na tela que se abre, *Fields Editor*, adicione todos os campos usando o botão direito do mouse e a opção *Add all fields*. Após isso, repita o processo no *cdsContas_Receber*.

Por fim, configure os eventos *AfterDelete*, *AfterInsert*, *AfterPost* e *BeforePost* do *cdsContas_Receber*, assim como fizemos com o *cdsContas_Pagar* no artigo anterior (que é apenas chamar o *ApplyUpdates* e configurar valores padrões para alguns campos). Como podemos ver não há grandes novidades em relação ao artigo anterior.

De fato todas essas configurações acabam tornando-se repetitivas, porém não há como fugir disso, pois as tabelas são semelhantes, tanto no desenvolvimento, quanto na carga de registros recebidos ao longo da vida do sistema.

Listagem 4. Código dos botões sbpPesquisa e tbnVisualizarTodos

```
procedure TfrmContasReceber.spbPesquisaClick(Sender: TObject);
begin
  inherited;
  if (Trim(edtPesquisa.Text) = '') or (Length(edtPesquisa.Text) < 4) then
  begin
    MessageDlg('Digite ao menos 4 (quatro) letras para realizar a pesquisa.', mtInformation,
      [mbOk], 0);
    edtPesquisa.SetFocus;
  end
  else
  begin
    with dmdPrincipal do
    begin
      if not PesquisarContasReceber(cbxPesquisa.Items[cbxPesquisa.ItemIndex],
        edtPesquisa.Text, cbxTipoBusca.ItemIndex) then
        MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
    end;
  end;
end;

procedure TfrmContasReceber.tbnVisualizarTodosClick(Sender: TObject);
begin
  inherited;
  if MessageDlg('Essa operação poderá demorar alguns minutos dependendo do volume de dados.'
    + #13 + 'Deseja prosseguir?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    with dmdPrincipal do
      if not CarregarTodos(sqlContas_Receber, cdsContas_Receber, 'CONTAS_RECEBER',
        'CODIGO') then
        MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
end;
```

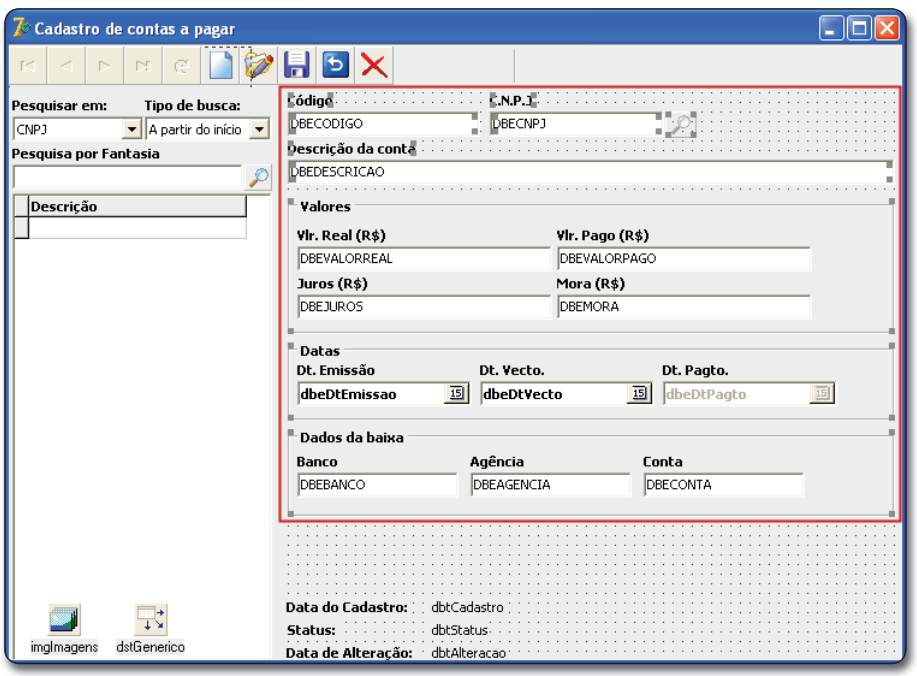


Figura 1. Copiando os componentes

Criando a tela de cadastro

Aqui também veremos poucas modificações. Criaremos uma nova tela de cadastro semelhante à janela de contas a pagar, modificando apenas a referência aos componentes de acesso no Data Module. Seguindo o esquema de criação de formulários baseado em herança, crie um novo formulário usando a opção *File>New>Other*. Selecione a aba *Forms* ou *SysPague* e em seguida clique em *frmPadrao*.

Os passos para configuração da janela já conhecemos. É necessário codificar o *spbPesquisa* no *box* de pesquisa e o *tbnVisualizarTodos*. A **Listagem 4** mostra o código do evento *OnClick* de cada botão indicado anteriormente.

Em seguida, configure a propriedade *DataSet* do *dtsGenerico* apontando-o para o *cdsContas_Receber* presente no Data Module.

> **Nota:** Lembrando que será necessário incluir a unit *dmdPrincipal* ao *uses* do formulário de contas a receber. Para isso, utilize as teclas de atalho ALT+F11. Aproveite e inclua as units *uPrincipal* e *uPesquisaPadrao*.

Para que possamos visualizar, alterar e manipular os dados da tabela em questão, é necessário adicionar componentes visuais vinculados ao banco de dados. Recomendo abrir a tela de cadastro de contas a pagar, marcar todos os componentes do corpo da janela e copiá-los para a memória (**Figura 1**).

Em seguida, cole-os na janela de contas a receber. Como nossos componentes da

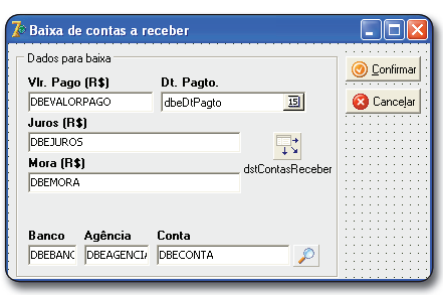


Figura 2. Diálogo de baixa de contas a receber

janela de Contas a Pagar são todos vinculados ao *dstGenerico*, padrão em todas as telas, ao colarmos na nova janela não é necessário refazer o vínculo de cada componente. Apenas revise as propriedades *DataSource* e *DataField* de cada *DBEdit* na intenção de verificar se todos estão corretamente ajustados.

No artigo anterior, criamos um pequeno botão ao lado do campo CNPJ que fazia a pesquisa de fornecedor e trazia o conteúdo do campo para a janela. Pois bem, faremos a mesma técnica nessa tela, porém utilizando a tabela CLIENTES. Modifique o *Name* do botão em questão para "sbbSelecionarCliente" e então digite o código da **Listagem 5** em seu evento *OnClick*.

Em contas a pagar, criamos um botão para baixar um registro em aberto. Em contas a receber também é necessário fazer a baixa do lançamento, pois é a partir das baixas que teremos condições de identificar os lançamentos que foram ou não pagos/recebidos. Tudo isso é importante pra ter um fluxo de caixa bem atualizado. Portanto, faça a inclusão de um novo botão na barra de botões da janela e codifique-o com o código da **Listagem 6**.

Observe que estamos fazendo referência ao *frmBaixaContasReceber*. O conceito é o mesmo da janela de contas a pagar, ou seja, crie um formulário idêntico ao *frmBaixarContasPagar*. Ela será usada para baixar o lançamento de contas a receber (**Figura 2**).

Como podemos ver, todo o processo de criação da janela de cadastro de contas a receber assemelha-se muito com o processo de criação efetuado no artigo anterior. Não há como errar, é como uma receita de bolo.

O importante é observar atentamente todos os passos e não esquecer de modificar os objetos e referências necessárias para vincular a janela à tabela de Contas a Receber.

Recurso especial

Como dissemos no início do artigo, criaremos um recurso interessante em nosso sistema que chamaremos de duplicação de lançamentos. A idéia é possibilitar que o usuário possa usar um registro no banco de dados como base para criar outros lançamentos do mesmo registro.

Por exemplo, ao adquirir um novo veículo financiado o usuário deverá incluir no sistema todas as parcelas de pagamento do veículo. Sem esse recurso será necessário clicar várias vezes no *Incluir* do formulário de cadastro e preencher praticamente os mesmos dados.

Com o recurso, basta o usuário informar o número de parcelas e a quantidade de dias entre uma e outra para que o sistema gere todos os lançamentos com sua devida data de vencimento. Vamos começar então desenhando uma nova janela, para isso clique em *File>New>Form* e salve-o como "uGerarRegistros.pas". Troque seu *Name* para "frmGerarRegistros".

A janela a ser criada é bastante simples em termos visuais. Basta adicionarmos dois *Labels*, dois *DBTexts* e dois botões ("btnConfirma" e "btnCancelar") na parte superior. Na parte inferior inclua dois *SpinEdits* ("speNumParcelas" e "speNumDias").

Configure a propriedade *Value* e *MinValue* de ambos para "1". Isso fará com que o valor mínimo ("*MinValue*") e o valor padrão ("*Value*") sejam 1 evitando erros futuramente. Pressione ALT+F11 e adicione o Data Module padrão ao *uses* do formulário em questão. Faremos agora as configurações necessárias para conexão com o banco de dados.

Inclua um *DataSoure* ("dstGenerico") e faça a ligação do mesmo ao *cdsContas_Receber* do Data Module. Em seguida selecione o primeiro *DBText* adicionado e configure sua propriedade *DataSource* e *DataField* escolhendo, respectivamente, *dstGenerico* e CNPJ.

Repita o último passo no *DBText* seguinte, porém no *DataField* escolha DESCRICAO. A explicação dessa configuração é simples: ao clicarmos no *tbnDuplicarLancamentos* que criaremos nas janelas de Contas a Pagar e Receber, chamaremos a caixa de diálogo em questão e o usuário poderá visualizar o CNPJ e a DESCRICAO do lançamento que será duplicado.

Ainda nesse botão, configuraremos a propriedade *DataSet* do *dstGenerico* apontando-o para *cdsContas_Pagar* ou *cdsContas_Receber* dependendo da ocasião. Nossa janela deverá se parecer com a **Figura 3**.

Listagem 5. Código do sbbSelecionarCliente

```
uses uPesquisaPadrao;
...
procedure TfrmContasReceber.sbbSelecionarClienteClick(Sender: TObject);
begin
  inherited;
  frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
  with frmPesquisaPadrao do
  begin
    FTabela := 'Clientes';
    FDataSetAlvo := dmdPrincipal.cdsClientes;
    ShowModal;
    if (ModalResult = mrOk) and not
      (dstGenerico.DataSet.IsEmpty) then
      frmContasReceber.dstGenerico.DataSet.FieldByName(
        'CNPJ').AsString := dstGenerico.DataSet.FieldByName('CNPJ').AsString;
  end;
  FreeAndNil(frmPesquisaPadrao);
end;
```

Listagem 6. Código do Baixar Contas a Receber

```
procedure TfrmContasReceber.tbnBaixaClick(Sender: TObject);
begin
  inherited;
  with dmdPrincipal, cdsContas_Receber, frmBaixaContasReceber do
  begin
    if not (FieldByName('STATUS').AsString = 'B') then
    begin
      frmBaixaContasReceber := TfrmBaixaContasReceber.Create(Self);
      Edit;
      dstContasReceber.DataSet.FieldByName(
        'VLR_PAGO').AsFloat := dstGenerico.DataSet.FieldByName('VLR_REAL').AsFloat;
      dstContasReceber.DataSet.FieldByName('DT_PAGTO').AsDateTime := Now;
      frmBaixaContasReceber.ShowModal;
      if frmBaixaContasReceber.ModalResult = mrOk then
      begin
        FieldByName('STATUS').AsString := 'B';
        Post;
      end;
    end
    else
      MessageDlg('Lançamento já baixado!', mtInformation, [mbOk], 0);
  end;
end;
```

Codificando a janela

Começaremos a codificar o formulário, incluindo a variável *FDataSetAlvo* na seção *public*. Portanto, vá até o início da unit e inclua essa variável do tipo *TDataSet*, como a seguir:

```
FDataSetAlvo: TDataSet;
```

Aproveite e declare na seção *private*, a função *GerarRegistros* como mostrado:

```
function GerarRegistros(ANumParcelas,
  ANumDias: Integer): Boolean;
```

A função, fará todo o processo de criação automática dos lançamentos de contas a pagar ou receber, conforme o número de parcelas e dias, passado como parâmetro. Pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho da função. Digite o código da **Listagem 7**.

Na listagem anterior podemos ver elementos da programação que para muitos podem ser desconhecidos, por isso veremos detalhadamente o que a função faz. O principal elemento da função *GerarRegistros* é a transação. Como faremos a inclusão de um ou mais registros no banco de dados, devemos isolar toda ação e prever o que faremos com os registros caso ocorra um erro.

Esse isolamento é chamado de transação e é bastante utilizado em rotinas grandes e com muitas inclusões, exclusões e atualizações em uma ou mais tabelas do banco, ou seja, toda rotina realizada deve ser monitorada e confirmada (*Commit*) caso não ocorram erros ou retornada ao estado original (*Rollback*) quando erros forem detectados.

No início do código atribuímos o valor 1 à propriedade *TransactionID* da variável *Transacao*. Em seguida dissemos que o modo de isolamento é *xillREADCOMMITED*. Feito isso, iniciamos a transação usando o *StartTransaction* do *sqlConexao* presente no Data Module, passando como parâmetro a variável de transação.

Isso significa que daqui pra frente toda a ação realizada está sendo "monitorada" e "gravada" pela transação. A seguir, atribuímos valores às variáveis CNPJ, DESCRICAO e VLR_REAL. Essas variáveis são usadas para duplicar o registro atual.

Iniciamos um *for* que vai de 1 até o número de parcelas indicadas pelo usuário no parâmetro *ANumParcelas*. Dentro do laço criamos um novo código para cada parcela incluindo ANO, MÊS e DIA no formato "AAAAMMDD" mais um número aleatório de três dígitos. Isso é necessário para evitarmos que o código se repita violando a chave primária da tabela. Por fim então, incluímos o registro.

Note que a data de vencimento do lançamento é gerada somando *Now* (data atual) mais o número de dias multiplicado pelo índice "i" do laço. Isso faz com que a data seja incrementada conforme o número de dias de intervalo determinado pelo usuário no parâmetro *ANumDias*.

Ao final do *loop* chamamos o *Commit* do *sqlConexao* fazendo com que todo o processo seja confirmado e enviado ao banco de dados. Caso ocorra algum erro, o *RollBack* do *sqlConexao* será chamado, e todas as inclusões serão desfeitas.

A última providência a ser tomada é efetuar a chamada a essa função no evento *OnClick* do *btnConfirmar*. Veja no código a seguir, que primeiro perguntamos ao usuário se ele deseja realmente confirmar a ação. Em caso positivo, chamamos a função passando como parâmetro a propriedade *Value* dos componentes *speNumParcelas* e *speNumDias* respectivamente:

```
if MessageDlg('Confirma procedimento?',
  mtConfirmation, [mbYes, mbNo], 0) = mrYes
then
  GerarRegistros(speNumParcelas.Value,
    speNumDias.Value);
```

Listagem 7. Código de geração de parcelas

```
uses DBXpress;
...
function TfrmGerarRegistros.GerarRegistros(ANumParcelas, ANumDias: Integer): Boolean;
var
  Transacao: TTransactionDesc;
  I: Integer;
  Codigo: string;
  Dias: Integer;
  CNPJ: string;
  DESCRICAO: string;
  VLR_REAL: Real;
begin
  Result := False;
  with dmdPrincipal do
  begin
    try
      Transacao.TransactionID := 1;
      Transacao.IsolationLevel := xilREADCOMMITTED;
      sqlConexao.StartTransaction(Transacao);
      Dias := 1;
      CNPJ := dstGenerico.DataSet.FieldByName('CNPJ').AsString;
      DESCRICAO := dstGenerico.DataSet.FieldByName('DESCRICAO').AsString;
      VLR_REAL := dstGenerico.DataSet.FieldByName('VLR_REAL').AsFloat;
      for I := 1 to ANumParcelas do
      begin
        Dias := I * ANumDias;
        with FDataSetAlvo do
        begin
          Randomize;
          Codigo := FormatDateTime('YYYYMMDD',
            FieldByName('DT_CADASTRO').AsDateTime) + FormatFloat('000', Random(999));
          Append;
          FieldByName('CODIGO').AsString := Codigo;
          FieldByName('CNPJ').AsString := CNPJ;
          FieldByName('DESCRICAO').AsString := DESCRICAO;
          FieldByName('VLR_REAL').AsFloat := VLR_REAL;
          FieldByName('DT_VECTO').AsDateTime := Now + Dias;
          FieldByName('STATUS').AsString := 'A';
          Post;
        end;
      end;
      sqlConexao.Commit(Transacao);
      MessageDlg('Processo efetuado com sucesso.', mtInformation, [mbOk], 0);
      Result := True;
    except
      on E:Exception do
      begin
        sqlConexao.RollBack(Transacao);
        MessageDlg('Ocorreu um erro neste processo.' +
          #13#13 + 'Erro gerado:' + #13 + E.Message +
          #13#13 + 'Todo o processo foi abortado!', mtError, [mbOk], 0);
        Result := False;
      end;
    end;
  end;
end;
```

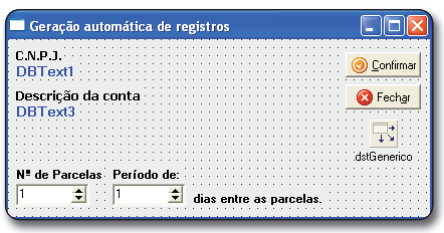


Figura 3. Diálogo de geração de registros

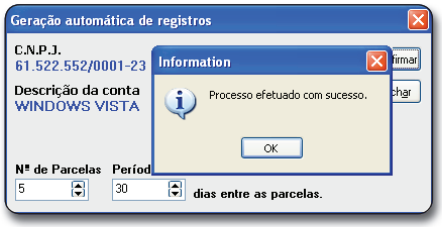


Figura 4. Sistema em execução

Listagem 8. Código do tbnGerarRegistros

```
procedure TfrmContasReceber.tbnGerarRegistrosClick(Sender: TObject);
begin
  inherited;
  if dstGenerico.DataSet.IsEmpty then
    MessageDlg('Nada selecionado.', mtInformation, [mbOk], 0)
  else
  begin
    frmGerarRegistros := TfrmGerarRegistros.Create(Self);
    with frmGerarRegistros do
    begin
      try
        FDataSetAlvo := dmdPrincipal.cdsContas_Receber;
        dstGenerico.DataSet := Self.dstGenerico.DataSet;
        ShowModal;
      finally
        FreeAndNil(frmGerarRegistros);
      end;
    end;
  end;
end;
```

Com isso finalizamos a tela de duplicação de lançamentos. Faremos agora, as chamadas à janela em questão nas telas de cadastro de Contas a Pagar e a Receber.

Alterando os cadastros

Inclua um novo botão ("tbnGerarRegistros") na barra de botões da janela Cadastro de Contas a Receber. No evento *OnClick* do botão inclua o código da **Listagem 8**.

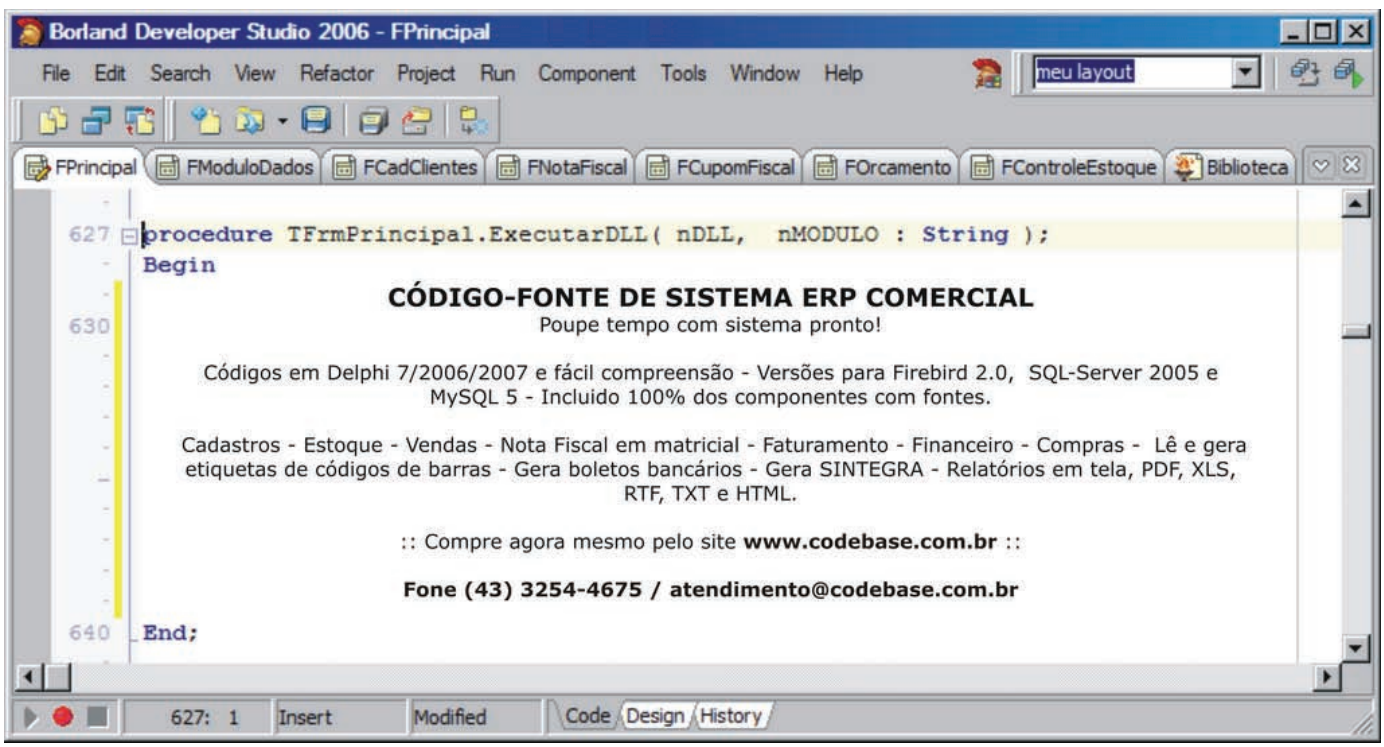
No evento, apenas verificamos se há um lançamento selecionado e então chamamos a caixa de diálogo. O segredo aqui é que configuramos a variável *FDataSetAlvo* criada na caixa de diálogo *frmGerarRegistros* informando o *ClientDataSet* que ele fará uso, nesse caso o *cdsContas_Receber*.

Para a janela de contas a pagar, repita os passos descritos anteriormente mudando somente a atribuição da variável *FDataSetAlvo* para *cdsContas_Pagar*. Não se esqueça de adicionar o formulário *frmGerarRegistros* ao *uses* de ambas as telas usando o atalho ALT+F11 ou *File>Use unit*. Veja na **Figura 4** o sistema em execução.

> **Nota:** Não esqueça de criar um botão e a chamada da janela de Contas a Receber na tela Principal.

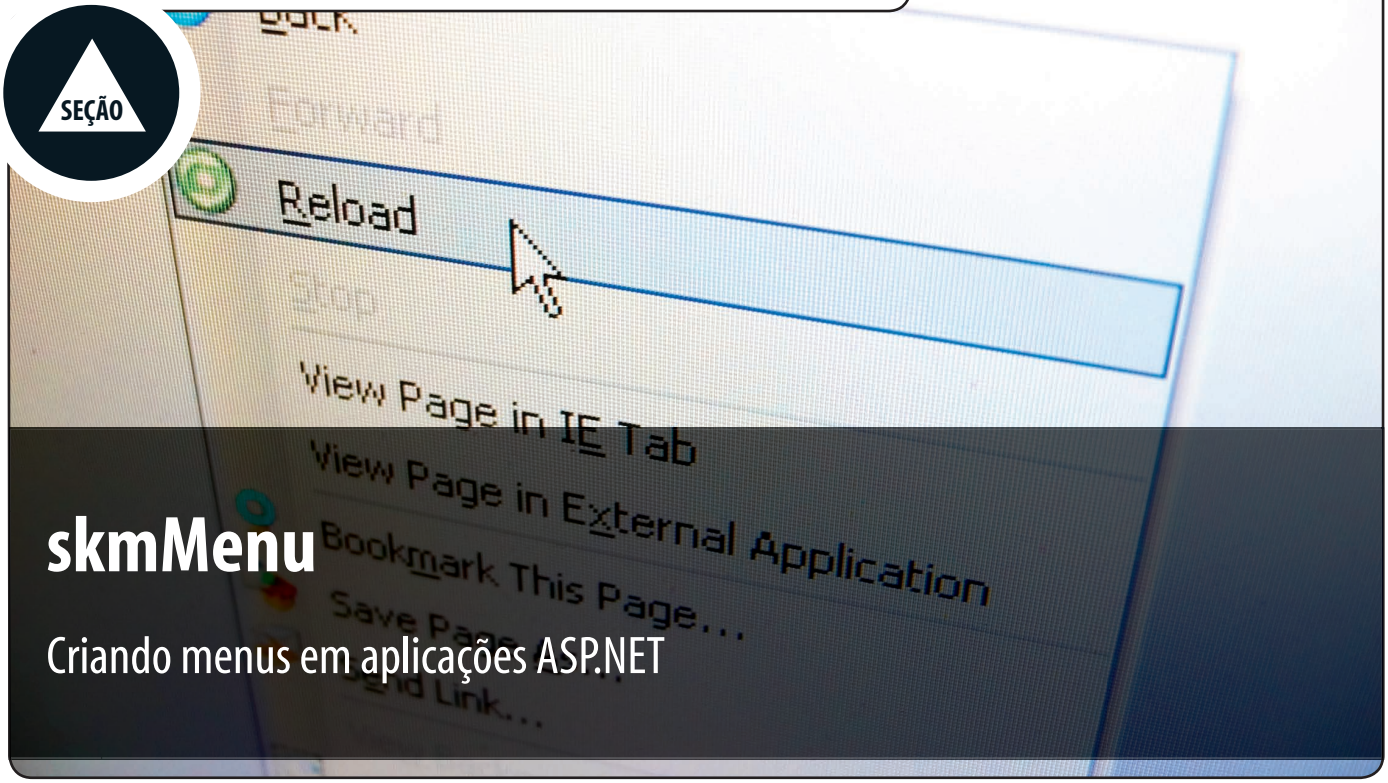
Conclusão

Nessa terceira parte do mini-curso do Sistema de Contas a Pagar e Cobrança vimos como criar uma janela de cadastro bem como o recurso especial de duplicação de lançamentos, facilitando ainda mais a vida do usuário na hora de incluir diversos lançamentos de uma só vez. Aproveito para agradecer ao leitor Francisco J. Riva pela dica e recomendação. Um forte abraço e até a próxima. ●



Artigos intermediários sobre Delphi Win32 e Delphi .NET

SEÇÃO



skmMenu

Criando menus em aplicações ASP.NET

Luís Fernando Quitaiski
(lfernandoq@gmail.com)
é Técnico em Informática formado pelo Colégio Frederico Jorge Logemann de Horizontina/RS. Desenvolvedor de aplicações Windows e ASP.NET com Delphi 2006 e Firebird.

Utilizar menus é uma forma muito prática e eficiente de organizar informações e opções dentro de uma aplicação. Neste artigo, ensinaremos como criar menus facilmente em aplicações ASP.NET com o Delphi 2006 através de um componente chamado *skmMenu*.

Abra o Delphi 2006 e crie um novo projeto através do menu *File>New>ASP.NET Web Application - Dephi for .NET*. O nome do projeto será "Menus" e utilizaremos o IIS como servidor Web. Criado o projeto, proceda então com a instalação do skmMenu.

O componente

O skmMenu é um componente *open source* utilizado para a criação dos menus em aplicações ASP.NET. Primeiramente, baixe o componente, através do site do projeto (*www.skmmenu.com*), usando o menu *Getting Started>Download*. Baixe a versão *Binary*, que já vem compilada. Até a data de criação deste artigo, a última versão era a 2.2. Para fazer o download, clique em *skmMenu Version 2.2 Binaries*. Salve o arquivo em algum local no seu computador e descompacte-o.

Feito isso, proceda agora com a instalação do componente no Delphi e sua adição na aplicação de exemplo. Para isso, no Delphi, vá até o menu *Components>Installed .NET Components* e na aba *.NET Components* clique em *Select an Assembly*. Na caixa que se abrirá, vá até o diretório onde o arquivo foi salvo e selecione o *skmMenu.dll*.

Agora, encontre o componente na lista e assegure-se que a caixa de seleção a sua frente está marcada, como na **Figura 1**. Caso essa caixa de seleção estiver desmarcada, o componente não aparecerá na paleta de componentes.

Feito isso, clique em OK. Nesse momento, na *Tool Palette*, haverá um componente chamado *Menu*, como pode-se observar na **Figura 2**. Esse é o componente que acabou de ser instalado e que será utilizado

para a criação dos menus. Clique no *Menu* e arraste-o para o formulário.

Principais propriedades

Veja agora as principais propriedades do componente, que visam definir, entre outras coisas, a sua aparência e o comportamento:

- *Layout*: define o *layout* horizontal ou vertical;
- *Font*: define o estilo da fonte que será utilizada nos menus, como nome da fonte, negrito, itálico e outros efeitos;
- *ForeColor*: define a cor da letra dos menus;
- *BackColor*: define a cor do fundo dos menus;
- *EnableViewState*: define se o estado do componente deve ser guardado entre uma requisição e outra;
- *ClickToOpen*: Define se, para abrir um menu, será necessário clicar sobre ele (*True*) ou somente passar com o mouse (*False*);
- ID: Define o nome do componente.

Destaca-se ainda a propriedade *SelectedMenuItemStyle*, onde podemos definir uma aparência diferente para o item que o usuário está apontando com o mouse no momento, diferenciando-o dos outros itens do menu. Isso cria um efeito muito interessante, semelhante ao produzido pelos menus utilizados em aplicações para Windows. Para este exemplo, configure as propriedades do componente da seguinte maneira:

- *BackColor*: "#E0E0E0";
- *Font>Name*: Arial;
- *EnableViewState*: *False* (já que não será necessário guardar o estado do componente);
- *Layout*: *Horizontal*;
- *SelectedMenuItemStyle>BackColor*: "#FFC080".

Para efeitos didáticos, adicione ainda um *RadioButtonList* e altere a propriedade ID para "rbTipodeUsuario", *AutoPostBack* para *True* e *EnableViewState* para *False*. Acesse a propriedade *Items* e aparecerá a caixa que permite que sejam adicionados itens ao *RadioButtonList*.

Clique em *Add* para adicionar um item e altere sua propriedade *Selected*, *Text* e *Value* para, respectivamente, *True*, "Administrador" e "Administrador". Adicione outro item clicando no *Add* e defina suas propriedades para, respectivamente, *False*, "Usuário" e "Usuario". Adicione também um *Label* e altere sua propriedade *Text* para "1". Seguindo todos esses passos, a aparência da página deverá ser parecida com a **Figura 3**.

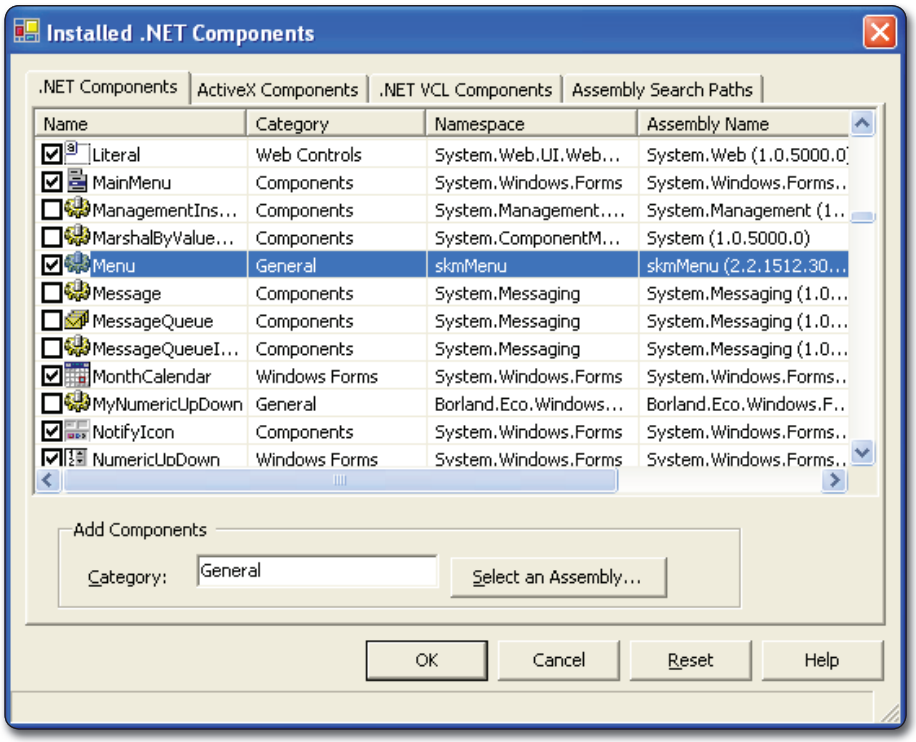


Figura 1. Instalação do skmMenu

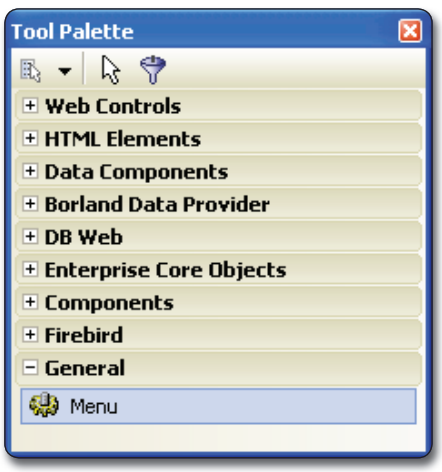


Figura 2. Componente instalado na Tool Palette

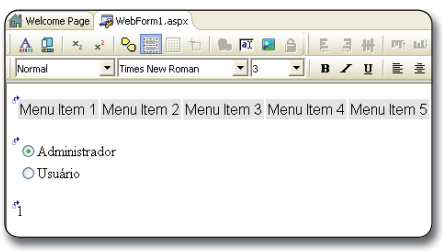
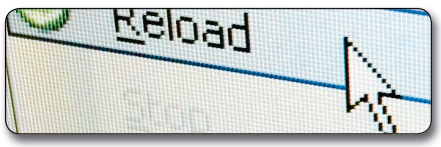


Figura 3. Aparência da aplicação após inseridos os componentes



Adicione mais uma página ao projeto, clicando no menu *File>New>Other* e na caixa que aparecer, dentro da categoria *Delphi for .NET Projects*, acesse *New ASP.NET Files*. Selecione *New ASP.NET Page* e dê um OK.

Nessa página adicione o *Menu* e configure as suas propriedades para os mesmos valores que foram escolhidos anteriormente (você pode copiar e colar o componente). Dê um ENTER após o componente para ir para a próxima linha da página e digite "Página 2".

Criando a estrutura dos menus

Configurada a aparência do *Menu*, será criado agora a sua estrutura, ou seja, os seus itens e submenus. Essas informações são armazenadas em um arquivo XML guardado dentro do diretório da aplicação.

Como um arquivo XML é composto apenas por texto, para criá-lo podemos utilizar qualquer programa que trabalhe com textos simples, como o próprio Bloco

de Notas do Windows. Neste artigo, será utilizado o Notepad++ (um Bloco de Notas "turbinado", disponível em *notepad-plus.sourceforge.net*) para a criação desse arquivo (**Figura 4**).

Abra, então, um editor de texto simples e salve, no diretório de sua aplicação (geralmente em *C:\Inetpub\wwwroot\Menus*), um arquivo chamado "menu.xml". Lembre-se que um arquivo XML é *case sensitive*, ou seja, há diferenças entre uma palavra escrita com letras maiúsculas e com letras minúsculas. Por exemplo: *text* é diferente de *Text*.

Em um arquivo XML, a primeira linha é uma identificação, que guarda a versão do XML que foi utilizada (nesse caso, a 1.0) e a codificação utilizada pelos caracteres do texto (a codificação escolhida afeta como são interpretados os caracteres que possuem sinais gráficos, como os acentos e o cedilha, e ela depende das configurações do servidor).

Recomenda-se que sejam testados os valores *iso-8859-1* e *iso-8859-2* e que seja escolhido o que se comportar corretamente. A linha será a seguinte: *<?xml version="1.0" encoding="iso-8859-1"?>*. Logo após a linha de identificação, tem-se uma tag que deverá englobar todos os itens do menu. Essa tag chama-se *menu* e é aberta logo após a identificação do arquivo e é fechada na última linha do mesmo.

Para criar um novo menu, utilizamos a tag *menuItem*. Criada a tag *menuItem*, vem a parte mais interessante: a criação dos textos do menu e a configuração da ação que será executada quando o usuário clicar sobre aquele item. Toda essa configuração também é feita por tags, e as principais estão listadas a seguir:

- *text*: texto que aparecerá no menu para o usuário;
- *image*: permite que o menu mostre, ao invés de texto, a imagem especificada;
- *tooltip*: dica referente ao menu, que será mostrada ao usuário se ele mantiver o mouse sobre aquele menu por algum tempo;
- *url*: página para onde o usuário será direcionado ao clicar no menu selecionado. Pode ser uma página da própria aplicação ou ser uma página da internet;
- *javascriptcommand*: comando *JavaScript* que será executado quando o usuário clicar no menu selecionado;
- *commandname*: utilizado se quisermos que, ao clicar no menu, seja executada uma ação codificada no sistema. As outras tags citadas, como *url* ou *javascriptcommand* são executadas pelo browser do cliente, como ir para outra página ou executar um *JavaScript*. O conteúdo dessa tag será passado à aplicação, onde o programador poderá executar a ação desejada utilizando-a como um identificador;
- *leftimage*: adiciona a figura especificada à esquerda do texto;
- *rightimage*: adiciona a figura especificada à direita do texto;
- *roles*: os usuários terão acesso àquele menu, separados por vírgula e sem espaço entre eles.

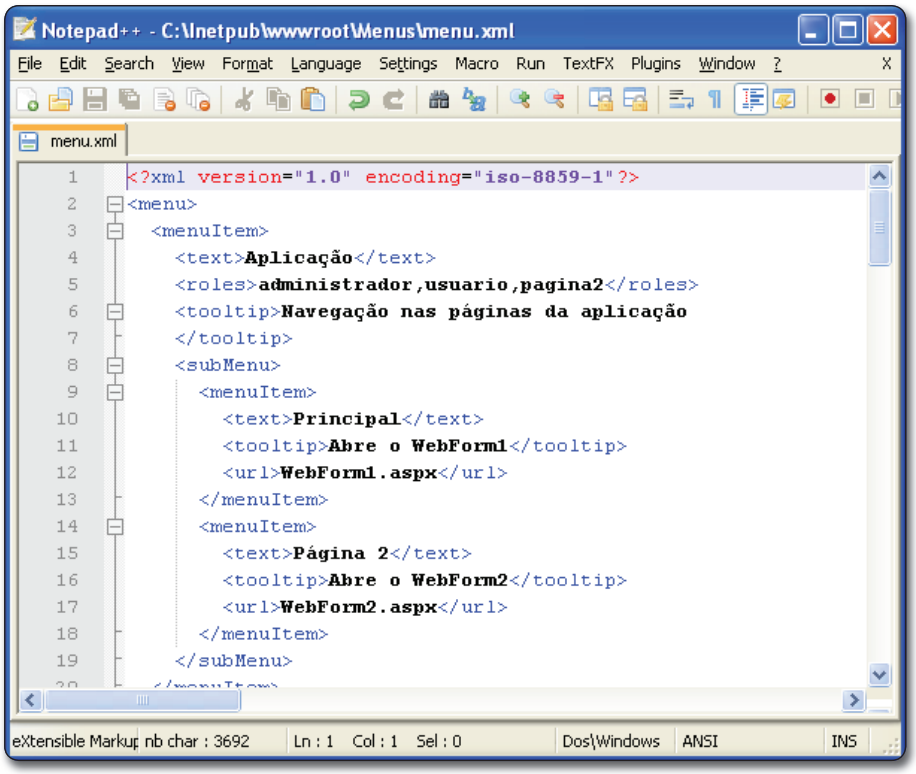


Figura 4. Exemplo de arquivo XML

> **Nota:** Se não forem especificadas as permissões de acesso para um submenu, elas serão herdadas do menu imediatamente superior.

Outra tag muito importante é a *subMenus*, que permite, como o nome sugere, que os menus possuam submenus, ou seja, que um menu surja quando o usuário passar com o mouse sobre outro menu. Seu uso é muito simples: abre-se a tag *subMenu*, cria-se quantos menus quisermos utilizando a tag *menuItem* e depois fechamos a tag *submenu*.

Para a criação de espaçadores existe a tag *menuSpacer*, onde podemos utilizar, dentro dela, as tags *spacermarkup* e *height*, utilizadas para, respectivamente, definir o caractere que aparecerá no espaçamento e definir o tamanho vertical da célula que conterá o caractere especificado pela tag *spacermarkup*. Na **Listagem 1** temos o XML utilizado nos menus do nosso exemplo.

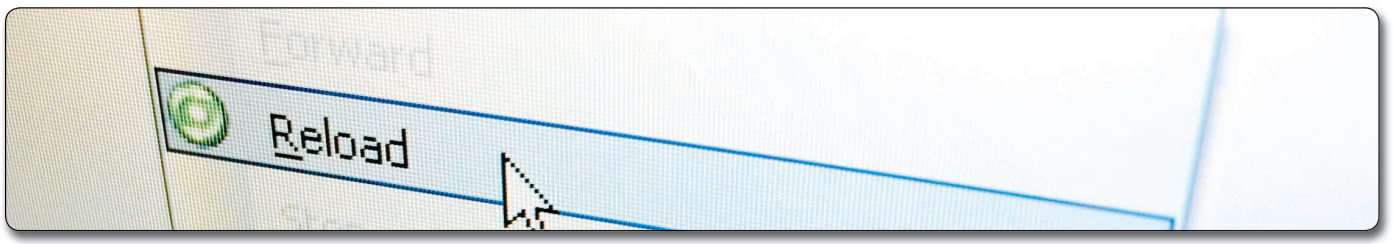
Codificações necessárias

Até esse momento, os menus já estão configurados e sua estrutura já está criada. Prossiga, então, colocando o código da **Listagem 2** no *Page_Load* do *WebForm1*.

Ele é responsável por passar o arquivo *menu.xml* para o componente que, baseado nas informações desse arquivo, criará os menus na tela. Ele também é responsável por passar ao componente, através do *UserRoles.Add*, os usuários que terão acesso àquele menu. Lembrando que esse valor pode ser obtido de qualquer lugar, como de uma tabela no banco de dados ou de uma variável de sessão, por exemplo.

Listagem 1. Conteúdo do arquivo menu.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<menu>
  <menuItem>
    <text>Aplicação</text>
    <roles>administrador,usuario,pagina2</roles>
    <tooltip>Navegação nas páginas da aplicação
    </tooltip>

    <subMenu>
      <menuItem>
        <text>Principal</text>
        <tooltip>Abre o WebForm1</tooltip>
        <url>WebForm1.aspx</url>
      </menuItem>

      <menuItem>
        <text>Página 2</text>
        <tooltip>Abre o WebForm2</tooltip>
        <url>WebForm2.aspx</url>
      </menuItem>

    </subMenu>
  </menuItem>

  <menuItem>
    <text>Páginas</text>
    <roles>administrador,usuario,pagina2</roles>
    <tooltip>Todos conseguirão ver esse menu</tooltip>

    <subMenu>
      <menuItem>
        <text>DevMedia</text>
        <tooltip>Abre a página da Editora DevMedia
        </tooltip>
        <url>http://www.devmedia.com.br</url>
      </menuItem>

      <menuItem>
        <text>Google</text>
        <tooltip>Abre a página do Google</tooltip>
        <url>http://www.google.com.br</url>
      </menuItem>

    </subMenu>
  </menuItem>

  <menuItem>
    <text>Email</text>
    <roles>administrador,pagina2</roles>
    <tooltip>Somente administradores verão esse menu
    </tooltip>

    <subMenu>
      <menuItem>
        <text>Hotmail</text>
        <tooltip>Abre a página de login do Hotmail
        </tooltip>
        <url>http://www.hotmail.com</url>
      </menuItem>

      <menuItem>
        <text>GMail</text>
        <tooltip>Abre a página de login do GMail
        </tooltip>
        <url>http://www.gmail.com</url>
      </menuItem>

    </subMenu>
  </menuItem>

  <menuItem>
    <text>Outros</text>
    <roles>administrador,usuario</roles>
    <tooltip>Mostrar como utilizar comandos JavaScript
      e comandos que serão executados pela aplicação
    </tooltip>

    <subMenu>
      <menuItem>
        <text>Exemplos JavaScript</text>
        <tooltip>Exemplos de como utilizar comandos
          JavaScript em um menu</tooltip>

        <subMenu>
          <menuItem>
            <text>Caixa de Mensagem</text>
            <tooltip>Mostra uma caixa de mensagens na
              tela utilizando JavaScript</tooltip>
            <javascriptcommand>javascript:alert (
              'Você clicou no menu Caixa de Mensagem');
            </javascriptcommand>
          </menuItem>

          <menuItem>
            <text>Janela Pop-Up</text>
            <tooltip>Abre, em uma janela pop-up,
              a página do Google</tooltip>
            <javascriptcommand>javascript:window.open (
               'http://www.google.com.br', '',
               'width=640px, height=480px');
            </javascriptcommand>
          </menuItem>

        </subMenu>
      </menuItem>
        <menuSpacer>
          <spacermarkup>&lt;hr noshade /&gt;
          </spacermarkup>
          <height>4px</height>
        </menuSpacer>

      <menuItem>
        <text>Chamar comando na Aplicação</text>
        <tooltip>Exemplos de como executar comandos
          na aplicação através do menu</tooltip>
        <subMenu>

          <menuItem>
            <text>Adicionar 1</text>
            <tooltip>Adiciona 1 unidade ao número que
              está aparecendo na tela</tooltip>
            <commandname>AdicionarUm</commandname>
          </menuItem>

          <menuItem>
            <text>Alterar Cor</text>
            <tooltip>Alterar a cor do número que está
              aparecendo na tela</tooltip>
            <commandname>AlterarCor</commandname>
          </menuItem>

        </subMenu>
      </menuItem>
    </subMenu>
  </menuItem>
</menu>
```



Coloque agora, no *MenuItemClick* do *Menu1* o código presente na **Listagem 3**. Esse evento é chamado quando o usuário clica em um item do menu que possui a tag *commandname*. Através da identificação passada pelo *commandname*, podemos executar comandos no sistema.

Coloque o código a seguir no *Page_Load* do *WebForm2*. Sua função também é passar o arquivo *menu.xml* como *DataSource* para o componente:

```
{ Somente mostrar os menus que possuam a role
  "pagina2" }
Menu1.UserRoles.Add ('pagina2');
Menu1.DataSource:= Server.MapPath ('menu.
xml');
Menu1.DataBind;
```

Compile a aplicação e veja o menu em ação. Ao clicar em um dos itens do *RadioButton* pode-se ver a tag *roles* em funcionamento, permitindo que somente os usuários declarados na tag *roles* tenham acesso àquele menu.

No menu *Aplicação* temos disponíveis as páginas do nosso projeto, onde podemos ir para a página principal ou para a página 2. Em *Outros*, há exemplos do menu executando comandos *JavaScript* (mostrando uma caixa de mensagens e abrindo, em uma janela *popup*, a página do Google). e, também de como fazer para que um clique no menu resulte em uma ação da aplicação (no exemplo, incrementar em 1 unidade o número que aparece na tela ou alterar a sua cor de preto para vermelho ou vice-versa). A aplicação compilada deve ficar semelhante à **Figura 5**.

Conclusão

Vimos neste artigo que a criação de menus em aplicações ASP.NET através do smkMenu é muito simples. Tudo o que temos a fazer é configurar a sua aparência e criar um arquivo XML com a estrutura dos menus, tarefas relativamente fáceis de serem feitas e que geram um efeito visual muito legal, além de ser uma ótima maneira de organizar os recursos oferecidos pela sua aplicação. ●

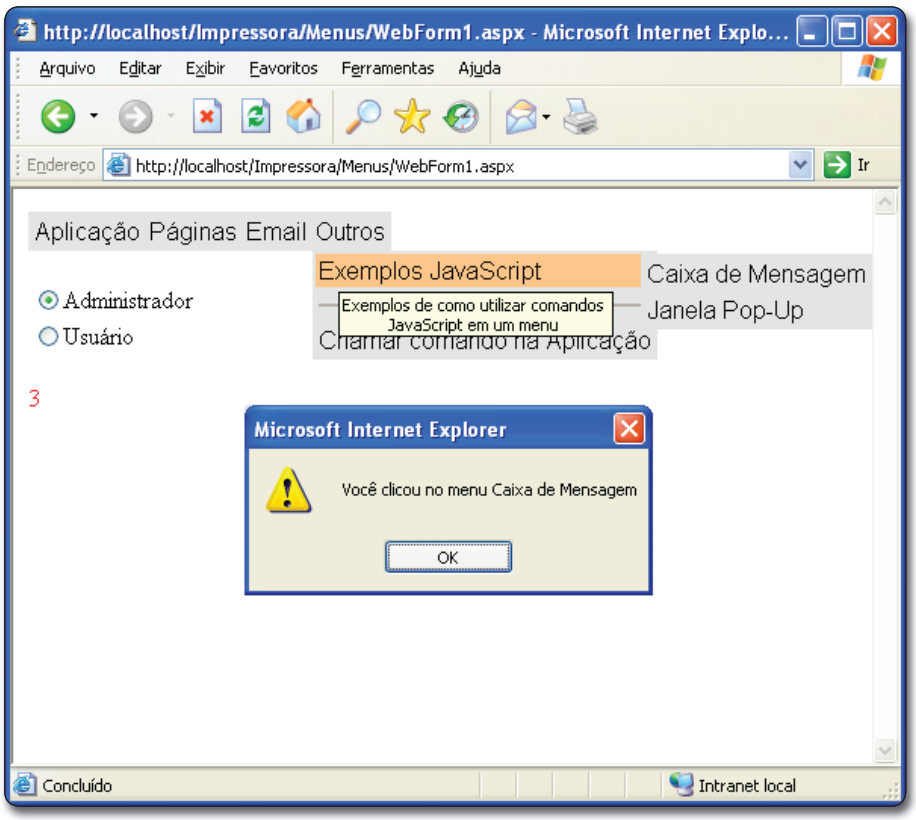


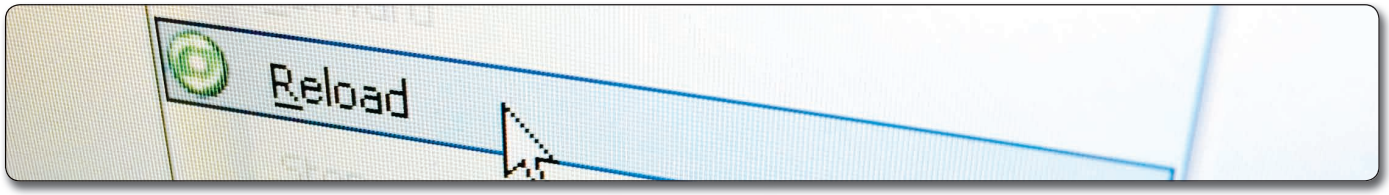
Figura 5. Aplicação em funcionamento

Listagem 2. Page_Load do WebForm1

```
{ Adiciona os usuários que terão permissão de
  acesso à esse menu }
Menu1.UserRoles.Add(rbTipodeUsuario.SelectedValue);
{ Passa o arquivo XML como DataSource para o menu e
  preenchê-lo }
Menu1.DataSource := Server.MapPath('menu.xml');
Menu1.DataBind;
```

Listagem 3. Evento MenuItemClick do Menu

```
{ Testa o menu clicado }

if e.CommandName= 'AdicionarUm' then
{ Comando executado se o item selecionado for
  AdicionarUm, então adiciona 1 unidade }
  Label1.Text:= Convert.ToString (Convert.ToInt16 (
    Label1.Text) + 1)
else

{ Comando executado se o item selecionado for
  AlterarCor }
begin
  { Verifica a cor atual e altera entre vermelho e
    preto }

  if Label1.ForeColor= Color.Red then
    Label1.ForeColor:= Color.Black
  else
    Label1.ForeColor:= Color.Red;
end;
```



Unimos a inovação e excelência da **Alog** com
o pioneirismo da **.comDomínio** para criar a
Alog Data Centers do Brasil:

LÍDER EM HOSTING GERENCIADO®

ALOG data centers + .comDominio DATA CENTERS

CONHEÇA OS NÚMEROS DESTA LIDERANÇA

2 Data Centers de Classe Mundial
200 Profissionais Dedicados
800 Clientes Corporativos
3.000 Servidores Hospedados
8.000 m² de Área Construída

www.alog.com.br | 0800 282 3330

ALOG DATA CENTERS DO BRASIL

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Artigos intermediários sobre Delphi Win32 e Delphi .NET

SEÇÃO

Crie um fórum de discussão

Usando Delphi 2006, ASP.NET e Firebird

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma win32 e .NET com banco de dados Firebird e MS SQL há 3 anos.

Neste artigo veremos uma forma simples de criar um site de fórum. Criaremos um espaço para discussão de assuntos e temas propostos pelos usuários do fórum. Começaremos fazendo a análise da estrutura de dados, após passaremos a criação do banco de dados. Utilizaremos o Firebird 1.5 e para desenvolvimento da aplicação o Delphi 2006 for ASP.NET.

Criaremos um fórum simples, com uma tela de listagem de áreas do fórum, e partindo-se da área selecionada, a listagem de todos os tópicos relacionados à mesma, e por fim o conteúdo de um tópico selecionado e a possibilidade dos usuários em responder ao tópico ou criar um novo.

Adotaremos neste exemplo uma regra de que a resposta e cadastro de tópicos, somente será realizada por usuários logados. Para fazer o controle de permissão de resposta e usuário logado, utilizaremos *User Controls* para rotinas de ocultar áreas de acesso restrito.

Nota: Devido ao pouco espaço disponível, alguns formulários não serão apresentados completamente. Faça o download dos fontes no endereço do artigo na página da DevMedia e acompanhe todas as funcionalidades do site de fórum.

Criando o banco de dados de exemplo

Para iniciarmos o exemplo, criaremos uma estrutura simplificada de tabelas e relacionamentos (**Figura 1**).

Após definida a estrutura, proceda com a criação do arquivo de dados. Crie sua base com o nome de "database.fdb". Com o banco de dados criado, passaremos agora a criação da aplicação no Delphi 2006, fique a vontade para utilizar também a versão do Delphi 2005 (o *script* do banco encontra-se para download).

Criando a aplicação

Crie uma nova *ASP.NET Web Application*

for Delphi .NET através do menu *File* do Delphi 2006, e nomeia a mesma como "forum". Utilizaremos neste artigo como servidor Web o *Internet Information Services* (IIS), se preferir poderá utilizar outros servidores compatíveis.

Pressione OK e aguarde para que o Delphi realize as configurações necessárias e crie a estrutura de páginas do projeto, onde você será direcionado à página *WebForm1.aspx*. Utilize o *Project Manager* do Delphi e renomeie a página para "index.aspx". Passaremos agora a criar a conexão da aplicação ao banco de dados criado anteriormente.

Configurando o componente de acesso a dados

Para realizar o acesso ao banco de dados Firebird faremos uso do provider, que deverá ser instalado no Delphi. Faça o download do componente no endereço *www.firebirdsql.org*. Localize no *Project Manager* a *Global.asax* e com um duplo clique ative a mesma, onde passaremos a configurar o componente de acesso.

Adicione na *Global.asax* um *FbConnection*, alterando o nome do mesmo para "conexao" e a propriedade *Modifers* para *Public*. Ao selecionar a propriedade *ConnectionString* uma caixa de diálogo será exibida, onde configuraremos as informações de acesso ao banco de dados (**Figura 2**). Informe o *Data Source* ou servidor do banco de dados, como "localhost" se a base estiver na mesma máquina.

Informe também o *Database* (caminho do banco). Se a senha do Firebird não foi alterada, teste sua conexão (*Test*) e finalize a configuração clicando em *Accept*. Acesse agora o código do arquivo utilizando a guia *Global.pas* na parte inferior da tela.

Localize entre o final da declaração de métodos e o início da declaração de *implementation*, e crie a variável "dados" do tipo *TGlobal*. Os demais componentes de consultas e parâmetros SQL serão criados em *runtime*, o que permite um maior controle das ações a serem executadas no decorrer da programação.

Customizando as páginas do fórum

Antes de retornarmos à *index.aspx*, criaremos uma unit para configurar códigos com consultas da aplicação. No menu *File>New>Other>Delphi for .NET Projects* na lista *New Files*, selecione *Unit* e confirme. Salve o projeto e atribua o nome de "UntGeral.pas" para a unit criada.

Criaremos na *UntGeral* uma *procedure*, que chamaremos no *Page_Load* de todas as páginas, chamada *VerificaConexao*, responsável por abrir o componente

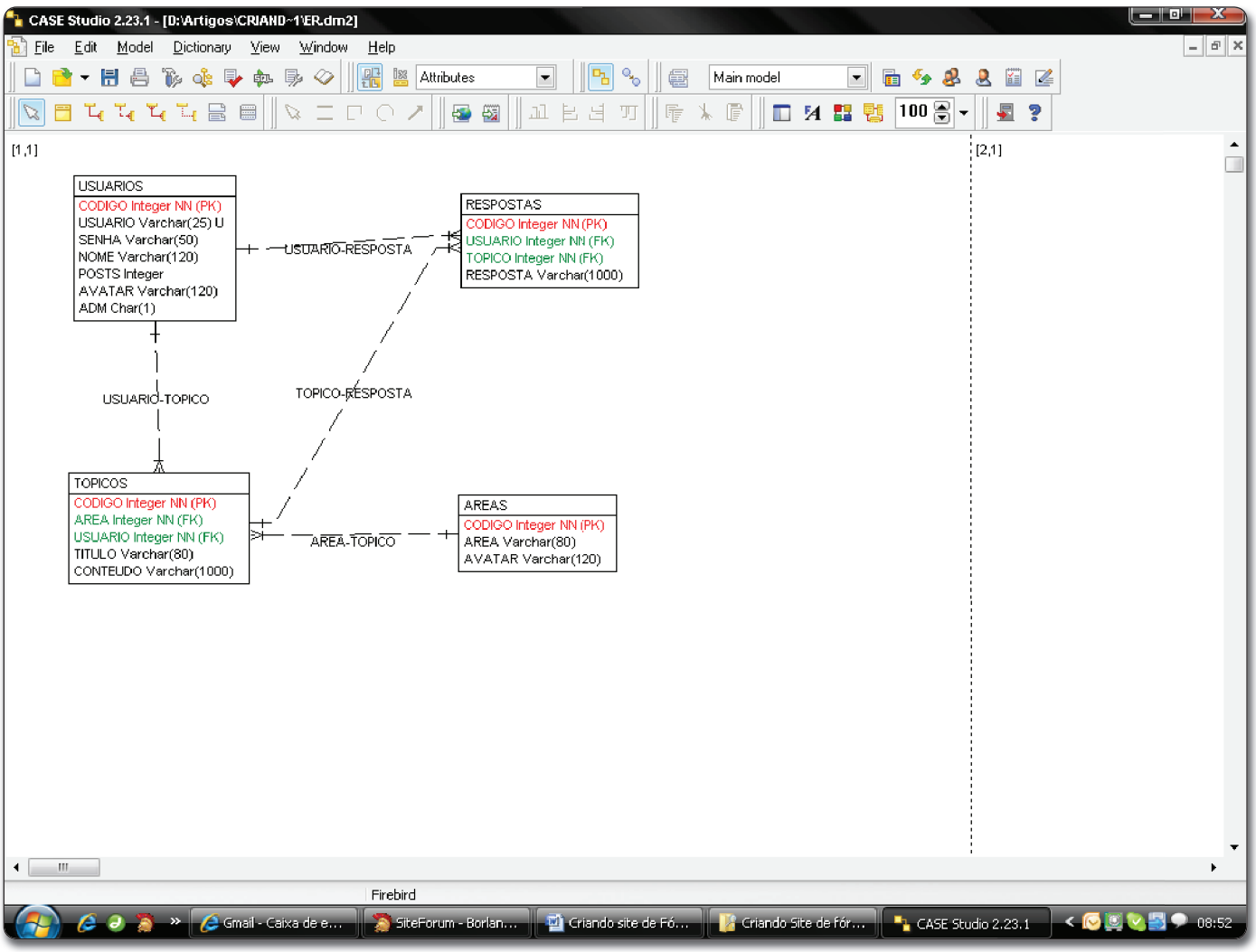


Figura 1. Diagrama ER da aplicação

de conexão a base de dados. Antes de codificarmos a mesma precisamos criar uma referência à *Global.asax*, onde colocamos o componente de acesso. Através do menu *File>Use unit* localize *Global* e confirme. Atribua o código da **Listagem 1** a *UntGeral*.

Na página principal (*index*), no *Page_Load* chamaremos a *procedure* recém criada (não esqueça de adicionar a unit *UntGeral*).

Controle de usuários na página

Criaremos um *User Control*, para adicionar o cabeçalho e apresentação da página, além de realizarmos o controle de permissões e usuários logados no site. No menu *File>New>Other* em *New ASP.NET Files* selecione *ASP.NET User Control* e confirme.

Altere o nome do *User Control* para "header.ascx". Adicionaremos na tela uma tabela para melhor organizar a disposição dos componentes. Adicionaremos também ao *User Control* os componentes que utilizaremos para controle dos usuários. Adicione dois *TextBox* ("TxtUsuario" e "TxtSenha"), e altere a propriedade *TextMode* do controle de senha para *Password*.

Adicione também um *Button*, altere o nome para "BtnLogin" e o *Text* para "Login". Adicione três *Labels* alterando o nome do primeiro para "LblUsuario", deixando a propriedade *Text* em branco e *Visible* para *False*.

No segundo, dê o nome de "LblUser" com o *Text* de "Usuário:" e no último, dê o nome de "LblSenha" com *Text* de "Senha:". Organize os componentes adicionados ao *User Control* conforme a **Figura 3**.

No *BtnLogin*, adicione o código da **Listagem 2**, que será responsável pela autenticação do usuário e senha no banco de dados. Adicione no *uses* os namespaces *FirebirdSQL.Data.Firebird* e *System.Web.Security*. Adicione também através do menu *File>Use Unit* a referência à *Global.asax*.

O código da listagem anterior é responsável por autenticar o usuário e senha na base de dados. Através de dois objetos criados (*FbCon* e *FbRead*), onde *FbCon* vinculado ao componente de conexão da *Global.asax* recebe um código SQL, que faz uma pesquisa de informações a partir dos parâmetros usuário e senha criados.

Perceba que o parâmetro *senha* recebe o resultado da criptografia do valor digitado pelo usuário (através do *FormsAuthentication.HashPasswordForStoringInConfigFile*). Em caso de sucesso, a SQL retorna informações que serão armazenadas em variáveis de sessão e será habilitada na página a descrição do usuário logado.

Pensando em utilizar esse *User Control* em todas as páginas do fórum, precisamos também no *Page_Load* realizar uma verificação para saber se o usuário já tenha passado pela autenticação, habilitando para os casos os campos de login e/ou a identificação do usuário logado. Adicione o código da **Listagem 3** ao *Page_Load* do *User Control*.

Nesse código, a presença do bloco *try.. except* é de extrema necessidade, pois em caso do usuário ainda não ter passado pela autenticação, as variáveis de sessão também não existiriam, o que provocaria um erro em tempo de execução.

Exibição das áreas do fórum – página inicial

De forma semelhante ao *User Control*, usaremos também uma tabela para organização dos componentes na página.

Listagem 1. Criando a VerificaConexao na UntGeral

```
procedure VerificaConexao;
begin
  if dados = nil then
  begin
    dados := TGlobal.Create;
    dados.conexao.Open;
  end;
end;
```

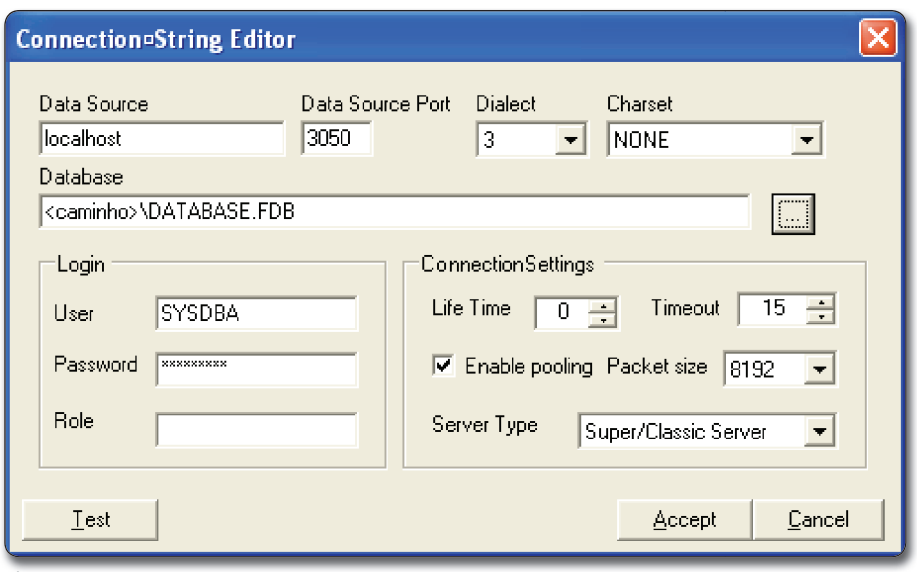


Figura 2. Configurando o componente de acesso a dados

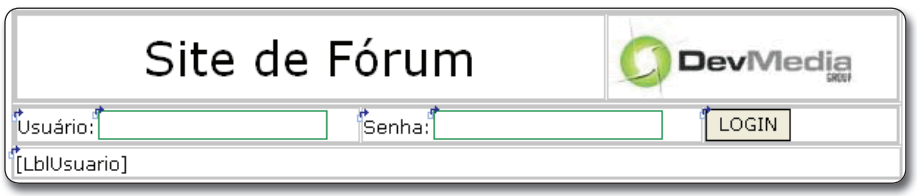


Figura 3. Organização dos componentes no User Control

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta, que mostra como salvar senhas de usuários com criptografia em aplicações ASP.NET.

www.devmedia.com.br/articles/viewcomp.asp?comp=1281

Nessa página, faremos uso apenas de um *DataGrid* que deverá ser renomeado para "GridAreas".

Na estrutura da tabela, teremos três linhas e uma coluna, onde na primeira arrastaremos o *User Control* do *Project Manager*. Na segunda adicionaremos o *GridAreas* e na última, os direitos do site (**Figura 4**). Fique a vontade para utilizar outros recursos de personalização e *layout* da página.

Para configurar o *GridAreas*, acesse o item *Property Builder* no rodapé do *Object Inspector*, onde uma caixa de diálogo será exibida. Clique na guia *Columns*, desmarque a opção *Create columns automatically at run time* e adicione quatro colunas conforme mostra a **Tabela 1**. Na guia *Format* configure a largura posição do texto das colunas.

Ainda na configuração da coluna onde será exibida a área, adicione na opção *URL field* o valor "CODIGO", que será carregado da base de dados. Em *URL format string* adicione "topicos.aspx?area={0}", que será o link a ser chamado quando o usuário clicar na linha do *DataGrid*, onde "topicos.aspx" é a página a ser exibida (criada mais adiante) e "area={0}" é o código da área onde serão listados os tópicos na página.

Voltando para as rotinas de código, criaremos agora na unit uma função que trará como resultado um *DataTable* contendo todas as áreas cadastradas no fórum. Através do *Project Manager* de um clique duplo sobre a unit e registre o seguinte código:

```
function ListaAreasForum: DataTable;
```

> **Nota:** Adicione após a declaração *interface*, uma nova seção *uses* e declare os namespaces *System.Data*, *FirebirdSQL.Data.Firebird*.

Implementando o código a nossa função, crie ela na seção *implementation* da unit digitando o código da **Listagem 4**.

No código anterior, observe que logo no início, criamos quatro objetos, que são utilizados para realizar a consulta no banco de dados (*FbCom* e *FbRead*) e também para montagem de uma lista dos resultados obtidos criando um *DataSource*. Após o *while*, onde preenchemos nossa

Listagem 2. Validando usuário e senha

```
procedure TWebUserControl1.BtnLogin_Click(sender: System.Object; e: System.EventArgs);
var
  FbCon: FbCommand;
  FbRead: FbDataReader;
begin
  try
    FbCon := FbCommand.Create('select usuarios.nome, usuarios.posts, '+
      'usuarios.avatar, usuarios.usuario, 'usuarios.codigo from usuarios where '+
      '((usuarios.usuario = ?) and (usuarios.senha = ?))', dados.CONEXAO);
    FbCon.Parameters.Clear;
    FbCon.Parameters.Add('USUARIO', DbType.&String);
    FbCon.Parameters.Add('SENHA', DbType.&String);
    FbCon.Parameters['USUARIO'].Value := TxtUsuario.Text.ToUpper;
    FbCon.Parameters['SENHA'].Value := FormsAuthentication.
      HashPasswordForStoringInConfigFile(TxtSenha.Text,'SHA1');
    FbRead := FbCon.ExecuteReader;
    if FbRead.Read then
    begin
      Session['USUARIO'] := FbRead['usuario'].ToString;
      Session['CODIGO'] := FbRead['codigo'].ToString;
      LblUsuario.Text := '<strong>' + FbRead['nome'].ToString +
        '</strong>, seja bem vindo ao site;';
      LblUsuario.Visible := true;
      TxtUsuario.Visible := false;
      TxtSenha.Visible := false;
      BtnLogin.Visible := false;
      LblUser.Visible := false;
      LblSenha.Visible := false;
    end;
  except
    LblUsuario.Text := 'Não foi possível efetuar login.';
    LblUsuario.Visible := true;
  end;
end;
```

Listagem 3. Page_Load do User Control

```
try
  if Session['NOME'].ToString <> '' then
  begin
    LblUsuario.Text    := '<strong>' + Session[
      'NOME'].ToString +
      '</strong>, seja bem vindo ao site;';
    LblUsuario.Visible := true;
    TxtUsuario.Visible := false;
    TxtSenha.Visible := false;
    BtnLogin.Visible := false;
    LblUser.Visible := false;
    LblSenha.Visible := false;
  end;
except
end;
```

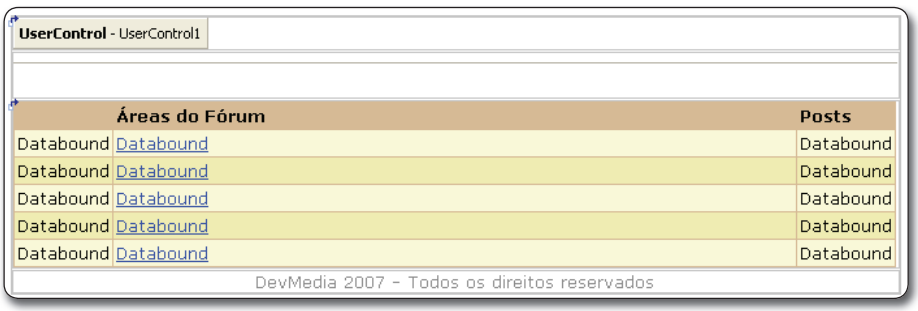


Figura 4. Organização dos objetos na página inicial

Tipo de Coluna	Texto de Cabeçalho	Campo de Dados	Visível
Bound Column	"Código"	CODIGO	Desmarque a opção
Bound Column		AVATAR	Deixe marcada
HyperLink Column	"Área"	AREA	Deixe marcada
Bound Column	"Tópicos"	TOPICOS	Deixe Marcada

Tabela 1. Configurando as colunas do DataGrid

Listagem 4. Função para listagem das áreas do fórum

```
function ListaAreasForum: DataTable;
var
  FbCom : FbCommand;
  FbRead: FbDataReader;
  Tabela: DataTable;
  Linha: DataRow;
begin
  Tabela := DataTable.Create;
  Tabela.Columns.Add('CODIGO', TypeOf(&String));
  Tabela.Columns.Add('AVATAR', TypeOf(&String));
  Tabela.Columns.Add('AREA', TypeOf(&String));
  Tabela.Columns.Add('TOPICOS', TypeOf(&String));
  FbCom := FbCommand.Create(
    'select areas.codigo, areas.area, areas.avatar, (select count(topicos.codigo) '+
    'from topicos where  topicos.area = areas.codigo) as topicos from areas',dados.CONEXAO);
  FbRead := FbCom.ExecuteReader;
  while FbRead.Read do
  begin
    Linha := Tabela.NewRow;
    Linha[0] := FbRead['codigo'].ToString;
    Linha[1] := '<img hspace="0"src="' + FbRead['avatar'].ToString +
      '" align="absMiddle" border="0">';
    Linha[2] := FbRead['area'].ToString;
    Linha[3] := FbRead['topicos'].ToString;
    Tabela.Rows.Add(Linha);
  end;
  Result := Tabela;
end;
```

Tipo de Coluna	Texto de Cabeçalho	Campo de Dados	Visível
Bound Column	Código	CODIGO	Desmarque a opção
HyperLink Column	Tópicos	TOPICO	Deixe marcada
Bound Column	Usuário	USUARIO	Deixe marcada
Bound Column	Resposta	RESPOSTA	Deixe Marcada

Tabela 2. Configurando as colunas do DataGrid de tópicos

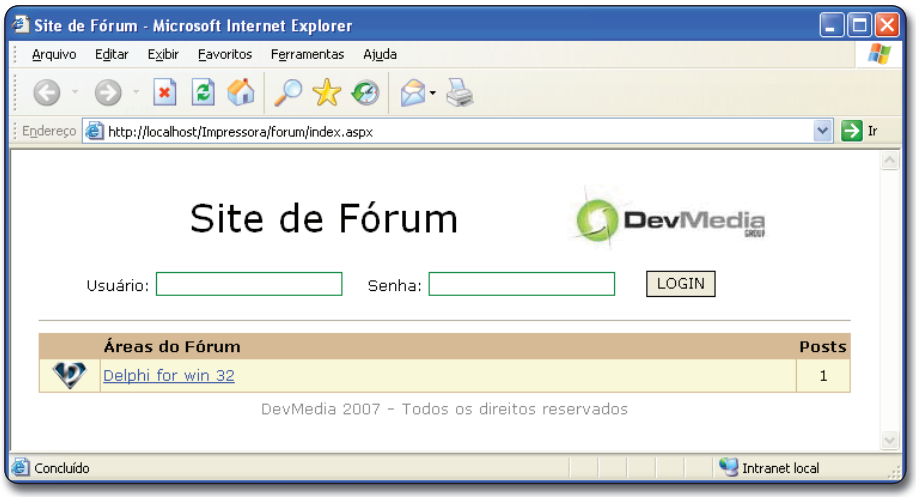


Figura 5. Aplicação em execução

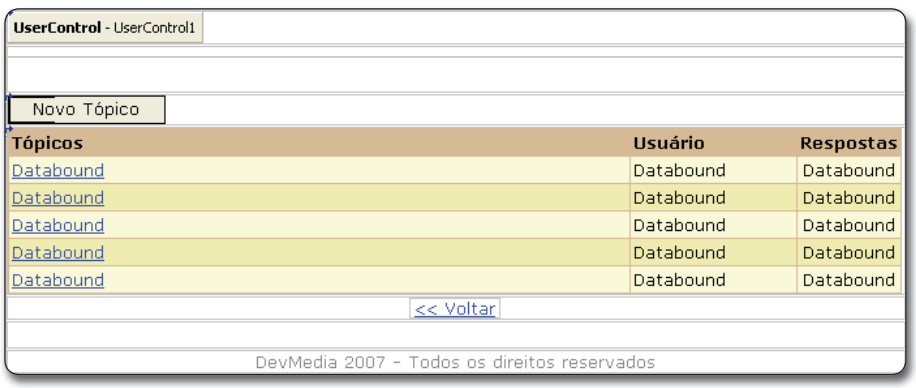


Figura 6. Organização dos objetos na página de exibição de tópicos

lista, jogamos o valor do objeto *Tabela* para o *Result*.

Para verificar o resultado da função, volte na página *index.aspx* e no *Page_Load* adicione o seguinte código, logo abaixo da *VerificaConexao*:

```
GridAreas.DataSource := ListaAreasForum;
GridAreas.DataBind;
```

Antes de executar sua aplicação, acesse o banco de dados e faça o cadastro de forma manual de uma área do fórum e após cadastre também um tópico qualquer relacionado à área cadastrada. Realizado o cadastro, compile e rode a aplicação para visualizar o resultado.

Veja agora que na estrutura da página foi carregada a área do fórum cadastrada com avatar e quantidade de *posts* a ela relacionados (**Figura 5**).

> **Nota:** Quando acessar o banco através do gerenciador de banco de dados para fazer os cadastros, crie antes os auto-incrementos das chaves primárias em todas as tabelas do banco. Como ainda não criamos nenhuma página de cadastro de usuários e por usar criptografia na validação da senha, cadastre um usuário através do gerenciador e informe como senha o valor criptografado *8AEFB06C426E07A0A671A1E2488B4858-D694A730*, que terá o valor *000* ao ser digitada na área de autenticação. Atribua seu *username* e grave o registro.

Definindo estrutura de exibição de tópicos

Nosso próximo passo é criar uma nova página que chamaremos de "topicos.aspx", configurando os componentes conforme mostra a **Figura 6**, para que nela sejam exibidas os tópicos referentes à área selecionada.

Configure o *DataGrid* através da criador de propriedades, conforme mostra a **Tabela 2** e altere seu nome para "GridTopicos". Para o *Voltar*, utilizamos um *HTML Label*, alterando o texto para "<< Voltar" e na propriedade *onclick* atribua o *javascript*: "*history.back();*"

Voltaremos agora para a unit de funções, onde criaremos uma consulta que retornará um *DataTable* carregado com

os tópicos referentes a área selecionada (**Listagem 5**).

Volte a *topicos.aspx* e no *Page_Load* da mesma, adicione o código a seguir:

```
VerificaConexao;
GridTopicos.DataSource :=
  ListaTopicosArea(Request.
  QueryString['area']);
GridTopicos.DataBind;
Session['CODAREA'] := Request.
  QueryString['area'];
```

Explicando o código anterior, na primeira linha estamos usando o código que verifica o status da conexão. Na segunda, estamos passando para a propriedade *DataSource* do *DataGrid* o resultado da consulta responsável pela listagem dos tópicos.

Como parâmetro para a função, estamos buscando o valor que mandamos junto ao endereço da página. Utilizamos o método *Request* de uma *QueryString*. Após, basta chamar o *DataBind* para que as informações do *DataGrid* sejam escritas na página.

Execute sua aplicação novamente e selecione a área do fórum, visualizando na próxima tela os tópicos referentes à área. Ao passar o mouse sobre um tópico, observe na barra que status que temos um link para exibição do conteúdo do tópico, que será a página que criaremos agora.

Exibindo tópicos e respectivas respostas

Adicione uma nova página ao projeto, salve-a como "leitura.aspx". Utilizaremos uma estrutura de componentes, um pouco diferente. Para mostrar as informações do tópico usaremos *Labels*, sendo um para o titulo ("LblTitulo"), um para exibir o nome do usuário que realizou o *post* ("LblUsuario") e também um *Label* para exibição do conteúdo do tópico ("LblConteudo").

Adicione também um *DataGrid* que usaremos para exibir as respostas que o tópico já obteve ("GridRespostas"), e organize os componentes da estrutura em tabelas, conforme mostra **Figura 7**.

Para exibir as informações do tópico na tela, criaremos novamente na unit um método que chamaremos de *ListaInformacoesTopico*, que receberá como parâmetro o código do tópico e resultará em três variáveis do tipo *string*, que serão os resultados a serem exibidos na página. Adicione o código da **Listagem 6** na implementação da *procedure*.

Listagem 5. Função para listagem das tópicos de uma área do fórum

```
function ListaTopicosArea(Area: string): DataTable;
var
  FbCom: FbCommand;
  FbRead: FbDataReader;
  Tabela: DataTable;
  Linha: DataRow;
begin
  Tabela := DataTable.Create;
  Tabela.Columns.Add('CODIGO', TypeOf(&String));
  Tabela.Columns.Add('TOPICO', TypeOf(&String));
  Tabela.Columns.Add('USUARIO', TypeOf(&String));
  Tabela.Columns.Add('RESPOSTA', TypeOf(&String));
  FbCom := FbCommand.Create(
    'select topicos.codigo, topicos.titulo, usuarios.usuario, (select '+
    'count(respostas.codigo) from respostas where respostas.topico = topicos.codigo) '+
    'as respostas from topicos inner join usuarios on (topicos.usuario = '+
    'usuarios.codigo) where ((topicos.area = ?))', dados.CONEXAO);
  FbCom.Parameters.Clear;
  FbCom.Parameters.Add('area', DbType.&String);
  FbCom.Parameters['area'].Value := Area;
  FbRead := FbCom.ExecuteReader;
  while FbRead.Read do
  begin
    Linha := Tabela.NewRow;
    Linha[0] := FbRead['codigo'].ToString;
    Linha[1] := FbRead['titulo'].ToString;
    Linha[2] := FbRead['usuario'].ToString;
    Linha[3] := FbRead['respostas'].ToString;
    Tabela.Rows.Add(Linha);
  end;
  Result := Tabela;
end;
```

Listagem 6. Procedure para listagem do conteúdo do tópico

```
procedure ListaInformacoesTopico(Codigo: string;
  var Titulo, Usuario, Conteudo: string);
var
  FbCom: FbCommand;
  FbRead: FbDataReader;
begin
  FbCom := FbCommand.Create(
    'select topicos.titulo, topicos.conteudo, usuarios.usuario, '+
    'usuarios.posts, usuarios.avatar from topicos inner join usuarios on (topicos.usuario = '+
    'usuarios.codigo) where ((topicos.codigo = ?))', dados.CONEXAO);
  FbCom.Parameters.Clear;
  FbCom.Parameters.Add('topico', DbType.&String);
  FbCom.Parameters['topico'].Value := Codigo;
  FbRead := FbCom.ExecuteReader;
  if FbRead.Read then
  begin
    Titulo := 'Assunto: <b>'+ FbRead['titulo'].ToString+'</b>';
    Usuario := '<img hspace="0" src="'+FbRead['avatar'].ToString+
      '" align="absMiddle" border="0" '+'WIDTH: 80px; HEIGHT: 100px>'+
      '<br><font face="Verdana" '+'style="font- size:11px">'+
      FbRead['usuario'].ToString+'</font><br>'+'<font face="Verdana" '+
      'style="font-size:10px">Posts: '+ FbRead['posts'].ToString+'</font>';
    Conteudo := FbRead['conteudo'].ToString;
  end;
end;
```

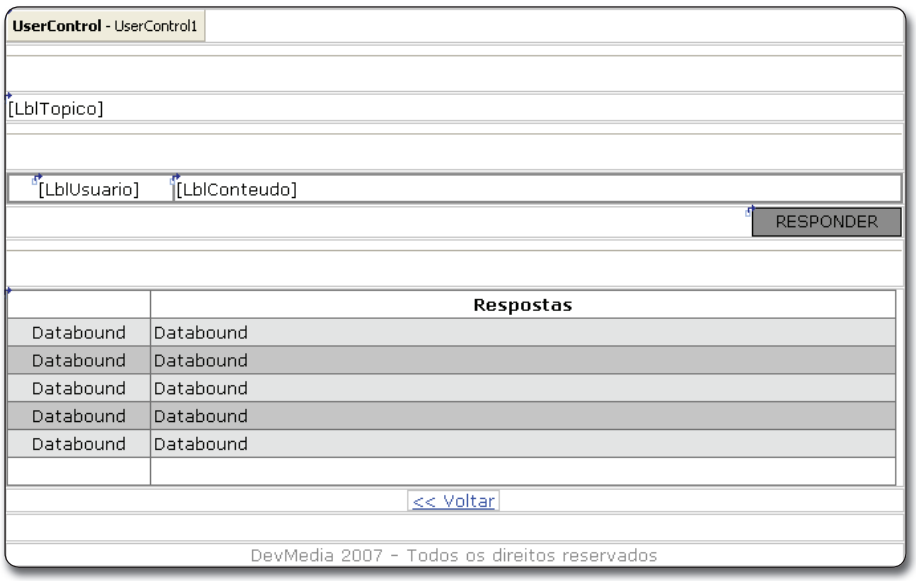


Figura 7. Organização dos objetos na página de leitura do tópico

Listagem 7. Page_Load da página de leitura do tópico

```
procedure TWebForm1.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  Codigo, Titulo, Usuario, Conteudo: string;
begin
  VerificaConexao;
  Codigo := Request.QueryString['topico'];
  ListaInformacoesTopico(Codigo, Titulo, Usuario, Conteudo);
  LblTopico.Text := Titulo;
  LblUsuario.Text := Usuario;
  LblConteudo.Text := Conteudo;
  Session['CODTOPICO'] := Codigo;
end;
```

Listagem 8. Implementação da ListaRespostasTopicos

```
function ListaRespostasTopicos(
  Codigo: string): DataTable;
var
  FbCom: FbCommand;
  FbRead: FbDataReader;
  Tabela: DataTable;
  Linha: DataRow;
begin
  Tabela := DataTable.Create;
  Tabela.Columns.Add('USUARIO', TypeOf(&String));
  Tabela.Columns.Add('RESPOSTA', TypeOf(&String));
  FbCom := FbCommand.Create(
    'SELECT RESPOSTAS.RESPOSTA, '+
    'USUARIOS.USUARIO, USUARIOS.AVATAR, '+
    'USUARIOS.POSTS FROM RESPOSTAS '+
    'INNER JOIN USUARIOS ON (RESPOSTAS.USUARIO = '+
    'USUARIOS.CODIGO) WHERE ((RESPOSTAS.TOPICO = ?))',
    dados.CONEXAO);
  FbCom.Parameters.Clear;
  FbCom.Parameters.Add('TOPICO', DbType.&String);
  FbCom.Parameters['TOPICO'].Value := Codigo;
  FbRead := FbCom.ExecuteReader;
  while FbRead.Read do
  begin
    Linha := Tabela.NewRow;
    Linha[0] := '<img hspace="0" src="'+
      FbRead['AVATAR'].ToString +
      '" align="absMiddle" '+
      'border="0" WIDTH: 80px; HEIGHT: 100px>'+
      '<br><font face="Verdana" '+
      'style="font-size:11px">'+
      FbRead['USUARIO'].ToString+
      '</font><br><font face="Verdana" '+
      'style="font-size:10px">'+
      'Posts: '+ FbRead['POSTS'].ToString+'</font>';
    Linha[1] := FbRead['RESPOSTA'].ToString;
    Tabela.Rows.Add(Linha);
  end;
  Result := Tabela;
end;
```

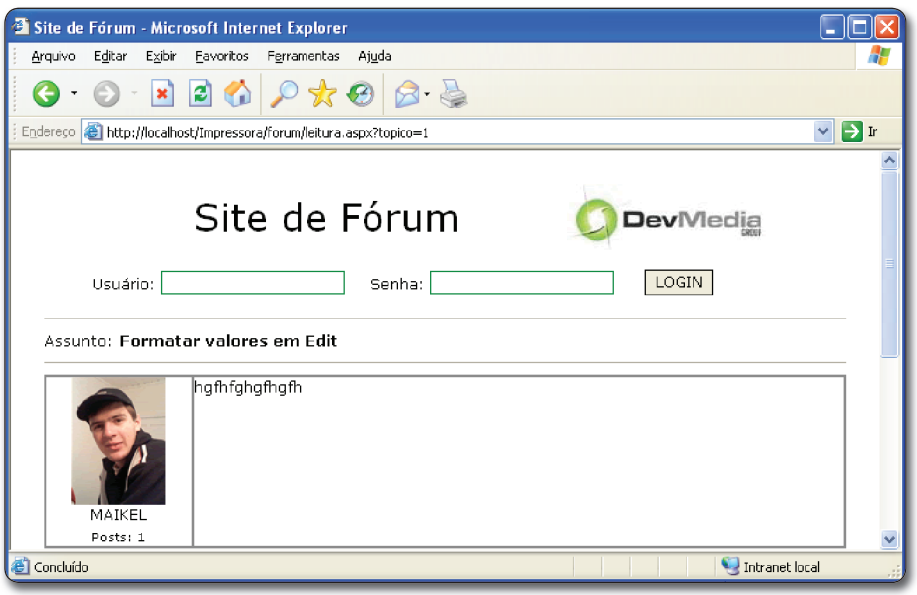


Figura 8. Visualizando tópicos e respostas

No código da listagem anterior, além de realizar uma consulta parametrizada e obter seus resultados, temos a formatação do conteúdo utilizando códigos *JavaScript* deixando os valores prontos para serem mostrados na página, no caso, onde as três variáveis da *procedure* estão recebendo valor.

Voltando para a *leitura.aspx*, onde mostraremos os resultados do procedimento na tela, atribua o código da **Listagem 7** no *Page_Load*, lembrando que precisa também criar as três variáveis que serão passadas à *procedure* e uma variável que utilizaremos para armazenar o *Request* do código do tópico.

Observe que primeiramente passamos as variáveis a *procedure* e depois quando estão carregadas com as informações, são vinculadas em seus respectivos *Labels*. Se executar o sistema, poderá observar que o conteúdo do tópico será exibido na página, porém se houver alguma resposta ainda não será exibida.

Para exibir as respostas, criaremos uma consulta, que receberá como parâmetro o código do tópico e retornará um *DataTable* carregado com os registros de resposta ao tópico. Crie o método com o nome de *ListaRespostasTopicos* e atribua o código da **Listagem 8**. Crie também no *GridRespostas* duas colunas, alterando as informações para "USUARIO" e "RESPOSTA".

No código anterior, muito semelhante aos já apresentados, estamos criando as colunas do *DataTable*, realizando a pesquisa no banco de dados e adicionando os resultados ao *DataTable*. Para exibir os resultados na tela, no *Page_Load* da página de leitura adicione o seguinte código:

```
GridRespostas.DataSource :=
  ListaRespostasTopicos(Codigo);
GridRespostas.DataBind;
```

Se executar a aplicação já será possível visualizar as respostas ao tópico sendo que as mesmas foram anteriormente cadastradas diretamente na base de dados (**Figura 8**).

Respondendo tópicos

Para possibilitar que o tópico seja respondido ou que novas respostas sejam adicionadas ao mesmo, criaremos uma nova página. Adicione um *Button* na tela

de leituras, alterando o nome para "BtnResposta" e a propriedade *Caption* para "Responder". Com um duplo clique sobre o mesmo (evento *Click*) adicione o código da **Listagem 9**, que verifica se o usuário está ou não logado.

Na nova página que acabamos de criar, altere o nome para "responder.aspx" e organize os componentes conforme a **Figura 9**.

No *Page_Load* adicione o código da **Listagem 10** que será responsável por verificar se o usuário está logado, para o caso do mesmo digitar a URL diretamente na barra de endereços no navegador e também fazer a verificação se o usuário possui algum tópico selecionado, além de verificar o status da conexão com a base de dados.

Perceba que não estamos fazendo nenhuma grande verificação nessa página, estamos apenas chamando as variáveis de sessão e testando se possuem algum valor. Caso alguma delas esteja em branco ou *nil*, quer dizer que ainda não foi criada, criando uma exceção, que será redirecionada para a página *index* da aplicação.

No botão de resposta do tópico, adicione ao evento *Click* o código da **Listagem 11** que será o responsável por cadastrar a resposta ao tópico selecionado, redirecionado após novamente para a listagem do tópico.

No código anterior, estamos criando novamente a variável *FbCom* (não se esqueça de declarar no *uses* o namespace *FirebirdSQL.Data.Firebird* e da página *Global.asax*), atribuímos o SQL de inserção do registro, criamos os parâmetros e após executamos o comando.

Agora que já temos um fórum montado, e que já estamos respondendo aos tópicos do mesmo, voltaremos até a *topicos.aspx* onde veremos como adicionar um novo tópico na área selecionada. No *BtnNovoTopico* que já adicionamos anteriormente, atribua o código da **Listagem 12**.

Listagem 9. Verifica se o usuário está ou não logado

```
try
  if Session['USUARIO'].ToString <> '' then
    Response.Redirect('responder.aspx');
except
  RegisterStartupScript('alerta',
    '<script>javascript:alert' +
    '(''Efetue Login'');</script>');
end;
```

Listagem 10. Verificando se o usuário esta logado

```
try
  VerificaConexao;
  if Session['USUARIO'].ToString <> '' then
  begin
    if Session['CODTOPICO'].ToString <> '' then
  end;
except
  Response.Redirect('index.aspx');
end;
```

Listagem 11. Código no botão de resposta de um tópico

```
procedure TWebForm1.Button1_Click(
  sender: System.Object; e: System.EventArgs);
var
  FbCom: FbCommand;
begin
  try
    FbCom := FbCommand.Create(
      'INSERT INTO RESPOSTAS (USUARIO, TOPICO, RESPOSTA) VALUES (?, ?, ?)', dados.CONEXAO);
    FbCom.Parameters.Clear;
    FbCom.Parameters.Add('USUARIO', DbType.&String);
    FbCom.Parameters.Add('TOPICO', DbType.&String);
    FbCom.Parameters.Add('RESPOSTA', DbType.&String);
    FbCom.Parameters['USUARIO'].Value := Session['CODIGO'].ToString;
    FbCom.Parameters['TOPICO'].Value := Session['CODTOPICO'].ToString;
    FbCom.Parameters['RESPOSTA'].Value := TxtResposta.Text;
    if FbCom.ExecuteNonQuery > 0 then
    begin
     Response.Redirect('leitura.aspx?topico=Session['CODTOPICO'].ToString);
    end;
  except
    on e:exception do
      RegisterStartupScript('erro', '<script>javascript:alert' +
        '(''Ocorreu o seguinte erro:\n'+ e.Message+''');</script>');
  end;
end;
```

Listagem 12. Adicionando um novo tópico

```
try
  if Session['USUARIO'].ToString <> '' then
    Response.Redirect('new_topico.aspx');
except
  RegisterStartupScript('alerta', '<script>'+
    'javascript:alert(''Efetue Login'');</script>');
end;
```

Cadastrando novos tópicos

Crie uma nova página e salve-a como "new_topico.aspx", que será utilizada para o cadastro de novos tópicos por usu-

Listagem 13. Page_Load da new_topico.aspx

```
try
  VerificaConexao;
  if Session['USUARIO'].ToString <> '' then
  begin
    if Session['CODAREA'].ToString <> '' then
  end;
except
  Response.Redirect('index.aspx');
end;
```

Listagem 14. Codificando o botão de cadastro de um novo tópico

```
procedure TWebForm1.BtnAddTopico_Click(
  sender: System.Object; e: System.EventArgs);
var
  FbCom: FbCommand;
begin
  try
    FbCom := FbCommand.Create(
      'INSERT INTO TOPICOS ( '+
      'AREA, USUARIO, TITULO, CONTEUDO) '+
      'VALUES (?, ?, ?, ?)',dados.CONEXAO);
    FbCom.Parameters.Clear;
    FbCom.Parameters.Add('AREA', DbType.&String);
    FbCom.Parameters.Add('USUARIO', DbType.&String);
    FbCom.Parameters.Add('TITULO', DbType.&String);
    FbCom.Parameters.Add('CONTEUDO', DbType.&String);
    FbCom.Parameters['AREA'].Value := Session['CODAREA'].ToString;
    FbCom.Parameters['USUARIO'].Value := Session['CODIGO'].ToString;
    FbCom.Parameters['TITULO'].Value := TxtAssunto.Text;
    FbCom.Parameters['CONTEUDO'].Value := TxtDescricao.Text;
    if FbCom.ExecuteNonQuery > 0 then
    begin
      Response.Redirect('topicos.aspx?area=Session['CODAREA'].ToString);
    end;
  except
    on e: Exception do
      RegisterStartupScript('erro', '<script>javascript:alert' +
        '(''Ocorreu o seguinte erro:\n' + e.Message+''');</script>');
  end;
end;
```

Figura 9. Organização dos objetos na página de respostas

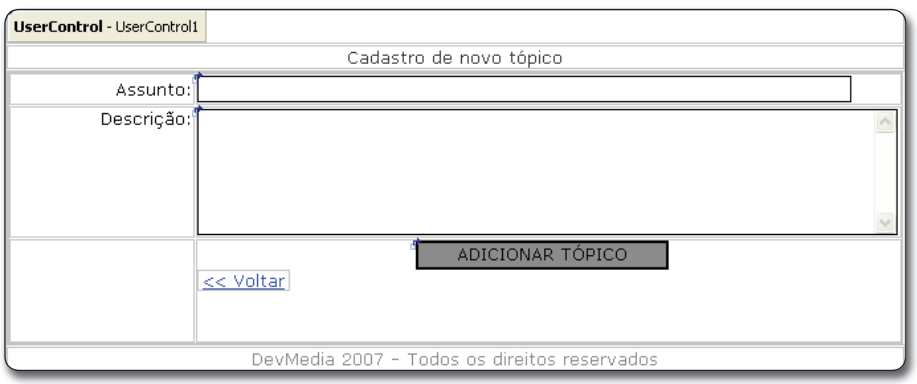


Figura 10. Organização dos objetos na página de novo tópico

ários. Atribua no *Page_Load* da mesma o código da **Listagem 13**, após configure os componentes na tela, conforme demonstrado na **Figura 10**.

Após definida toda a estrutura da página de cadastro de novos tópicos, faremos a codificação do botão responsável por cadastrar as informações do tópico, e para isso atribua ao evento *Click* do *BtnAddTopico* o código da **Listagem 14**.

Da mesma forma que nas outras páginas, declare o namespace *FirebirdSQL.Data.Firebird* e adicione referência à unit da *Global.asax* onde está configurada a conexão do projeto. Declaramos a variável *FbCom*, que será usada para a criação do comando SQL de inserção do registro.

Se o comando for executado corretamente, o usuário será redirecionado para a listagem do tópico cadastrado. Em caso de algum problema durante o cadastro, usaremos o tratamento de exceções para exibir a falha.

É interessante que você faça a verificação e validação dos campos, não permitindo que o usuário salve registros com campos sem informação. Para isso, você poderá usar os componentes de validação (*RequireFieldValidator, RegularExpressionValidator* etc), entre outras formas possíveis, ou também por controle manual.

> **Nota:** Neste exemplo não foi abordado o cadastro de usuários, bem como a manutenção dos mesmos, além da manutenção de áreas e tópicos do fórum, sendo muito simples de ser feito. Você poderá fazer o download completo deste exemplo no site da DevMedia.

Conclusão

Neste artigo vimos como desenvolver de forma simples um fórum de discussão, contemplando desde os controles de usuários, listagem de áreas, tópicos e respostas. Até a próxima. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta, que mostra como trabalhar com Validators no ASP.NET.

www.devmedia.com.br/articles/viewcomp.asp?comp=2341

Cursos Online

Assinatura .net magazine PLUS

Mais conteúdo .NET por muito menos!

A revista **.net Magazine** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade .
Conheça abaixo os cursos já disponíveis.

Curso em destaque

Crie uma loja virtual completa

Confira neste curso Online como construir uma loja virtual completa no Visual studio 2005. Aprenda como criar o banco de dados, carrinho de compras, profile, página de erros e muito mais .

Confira o plano de aula completo:
www.devmedia.com.br/lojavirtual

A sua melhor opção de aprendizagem!
Assine a **.net Magazine** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: www.devmedia.com.br/curso

- Construindo relatórios com Crystal Reports e Visual Studio 2005
- Criando uma aplicação Web Completa
- Criando uma aplicação client/server no Visual Studio 2005
- Aprenda a criar um blog com ASP.NET
- Curso de C#
  * Curso em andamento

DevMedia GROUP Mais Informações: www.devmedia.com.br/central - Tel.: 2220-5375 / 2220-5435

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Cadastros simples

Crie um cadastro de Departamentos e Funcionários com banco de dados local

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Bem-vindos a nova coluna da revista ClubeDelphi: *Easy Delphi*. Essa coluna é destinada aos leitores iniciantes que estão começando o aprendizado das técnicas de programação e desejam conhecer melhor a ferramenta e os conceitos básicos para desenvolver sistemas.

O objetivo é mostrar aplicações simples, com linguagem didática e detalhada de todo o processo de criação de software. Nesse primeiro artigo da coluna veremos como criar um projeto utilizando banco de dados local, ou seja, sem a necessidade de um servidor.

Faremos o desenvolvimento de um sistema onde cadastraremos Departamentos e Funcionários de uma empresa em modo master/detail que veremos mais adiante o significado.

O que é um banco de dados?

Antes de iniciarmos nosso sistema, precisamos primeiramente entender o significado e os conceitos básicos de banco de dados. Todo e qualquer sistema de consulta e cadastro necessita de um banco de dados onde as informações são gravadas para consulta posterior. A gravação desses dados é chamada de persistência de dados. As informações são enviadas para arquivos que normalmente ficam armazenados em servidores que disponibilizam tais informações para todos os usuários em uma rede.

Esse conceito é chamado *client/server* ou Cliente/Servidor, onde o desenvolvimento acontece em duas camadas. No servidor ficam os arquivos do banco de dados e nas estações de trabalho, chamadas clientes, fica o software que acessa o servidor e conseqüentemente o banco de dados.

Nesses arquivos são criadas tabelas que na verdade são conjuntos de linhas e colunas responsáveis pela entrada de dados em um banco. Um exemplo disso seria uma tabela de clientes onde terí-

amos as colunas: Nome, Razão Social, CNPJ, Endereço, Cidade, Estado, CEP e Telefone.

Cada coluna em uma tabela também é chamada de *Campo* ou *Field*. É importante entendermos também que cada informação gravada exige um tipo de dado diferente. Existem diversos tipos de dados e eles são usados para armazenar corretamente cada informação. Vejamos alguns tipos de dados:

- *VarChar* ou *String*: armazena caracteres alfa, ou seja, letras, números e caracteres especiais. Normalmente é utilizado para guardar nomes, telefones, e-mails etc.;
- *Numeric*, *Float*, *Double Precision*: são usados para armazenar valores numéricos com decimais. Salários, preços de produtos, taxas, juros, moras, enfim, valores monetários;
- *Date*, *Time* e *TimeStamp*: são tipos que armazenam Data, Hora ou Data e Hora respectivamente. Normalmente usados para data de nascimento, horário de entrada e saída de mercadorias em estoque ou mesmo data de emissão de notas ficais, por exemplo;
- *Integer*: guarda valores inteiros, ou seja, somente números sem o uso de decimais.

Há outros tipos de dados, porém esses são os mais comuns e utilizados no dia-a-dia. Um banco de dados pode conter diversas tabelas cada uma com sua finalidade. Outro fator importante em um programa é a relação master/detail entre as tabelas de um banco.

Dizemos que um relacionamento é Master/Detail (mestre/detalhe) quando um registro em uma tabela está vinculado a outro registro de outra tabela, como podemos ver no exemplo da **Figura 1**.

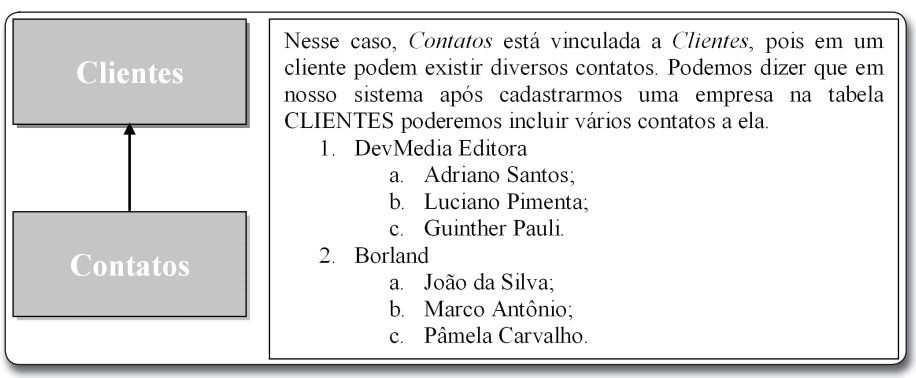


Figura 1. Exemplo para um relacionamento mestre/detalhe

Em nosso exemplo criaremos um banco de dados local. Isso significa que nossos arquivos de dados ficarão no mesmo local onde o sistema ficará instalado. Faremos o desenvolvimento de duas tabelas: DEPARTAMENTO e FUNCIONARIOS.

Desenhando o sistema

Para iniciar um novo projeto, abra o Delphi 7 e logo após clique no menu *File>New>Application*. Isso fará com que o Delphi crie uma nova janela onde colocaremos nossos componentes visuais e não-visuais para acesso e manutenção dos dados que serão gravados.

Salve o formulário com o nome de "uPrincipal.pas" usando o menu *File>Save As*. Em seguida salve o projeto com o nome de "Easy1.dpr", usando o menu *File>Save Project As*. A primeira providência a tomarmos antes de desenhar a janela principal, é fazer a criação do banco de dados e preparar o sistema pra acessá-lo. Utilizaremos um recurso bastante útil numa aplicação real que é a criação de um Data Module.

O Data Module é um componente do tipo container. Sua finalidade é armazenar todos os componentes de acesso a dados. É usado para centralizar e organizar melhor os componentes de acesso, pois em um software com muitas janelas é comum cadastros diferentes acessarem os mesmos dados.

Um Data Module não fica visível pra o usuário final , pois é usado apenas internamente pelo sistema. Para criar um Data Module clique em *File>New>Data Module*. Uma pequena janela branca é exibida no Delphi e é nela que colocaremos os componentes (**Figura 2**).

Salve o Data Module como "udmPrincipal.pas" usando o menu *File>Save As*. Em seguida clique em uma área em branco do Data Module e mude seu *Name* para "dmPrincipal". A primeira tabela que criaremos no sistema é a de *Departamentos*, usando para isso, o *ClientDataSet* da paleta *Data Access*.

> **Nota**: Um ClientDataSet é uma forma de representar uma tabela de dados no Delphi.

Criando campos

Adicione um *ClientDataSet* no Data Module e em seguida mude sua propriedade *Name* para "cdsDepartamentos". Após isso, clique duas vezes nos componentes, para que uma pequena caixa de diálogo seja aberta. Chamamos essa janela de *Fields Editor* (editor de campos). É nela que criaremos todos os campos da tabela (**Figura 3**).

Para incluir os campos (colunas) da tabela, clique com o botão direito em uma área branca da caixa de diálogo e escolha *New Field* (novo campo). Ao clicar em *New Field* a janela de inclusão do campo será aberta (**Figura 4**).

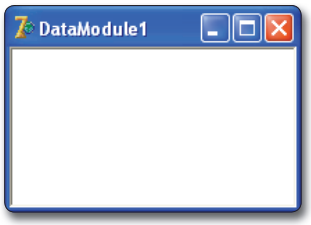


Figura 2. Criação do Data Module

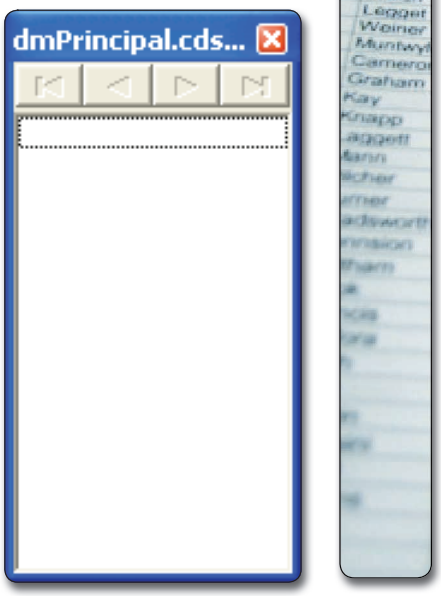


Figura 3. Editor de campos do ClientDataSet

O primeiro grupo de campos (*Field properties*) é usado para configurar as propriedades que são: *Name*, *Type* e *Size*. Digite "ID" e em seguida escolha o tipo *Autoinc* em *Type*. Isso cria um campo que tem o seu valor gerado automaticamente. Por fim, escolha a opção *Data* no grupo de propriedades *Field type* e confirme clicando em OK. Repita a ação incluindo dessa vez o campo "DEPARTAMENTO" do tipo *string* com o tamanho igual a "30".

Com isso, concluímos a construção da estrutura da tabela. Agora faremos com que o Delphi crie o arquivo físico no disco rígido. Para isso, clique com o botão direito no *cdsDepartamentos* e escolha a opção *Create Dataset*. Novamente clique com o botão direito e em seguida em *Save to MyBase Xml table*.

O Delphi abrirá uma caixa de diálogo solicitando o nome e a pasta onde o arquivo de banco de dados será criado. Selecione a pasta onde salvou os arquivos do projeto e dê o nome de "Departamentos.xml". Feito isso, vamos criar o segundo arquivo do sistema de departamentos e funcionários. Devemos agora incluir um novo *ClientDataSet* e repetir os passos efetuados anteriormente.

> **Nota:** Não esqueça de mudar a propriedade *Name* para "cdsFuncionarios".

Vamos incluir diversos campos na estrutura da tabela *Funcionarios*. Então, siga os passos anteriores e crie uma tabela com os campos informados na **Tabela 1**.

Campos calculados

Como podemos ver na tabela *Funcionarios* temos os campos VLR_HORA e DIAS_TRABALHADOS. Esses campos serão usados para calcular o salário mensal do funcionário, conforme a seguinte equação:

```
SALARIO = VLR_HORA * 8 * DIAS_TRABALHADOS
```

Estamos multiplicando o valor recebido por hora, por oito horas trabalhadas por dia, vezes a quantidade de dias trabalhados no mês. Dessa forma, obtemos o salário do funcionário. Para armazenar tal cálculo vamos criar um campo especial no banco chamado SALARIO e esse por sua vez receberá o valor já calculado.

Clique com o botão direito no *cdsFuncionarios* e selecione *New field*. Digite "SALARIO" em *Name* e escolha o tipo *Float*. Em seguida escolha a opção *Calculated* e confirme em OK (**Figura 5)**.

Agora basta dar um *Create Dataset* e em seguida mandar criar o arquivo XML, usando a opção *Save to MyBase Xml table*. Salve o arquivo com o nome de "Funcionarios.xml". O formato XML é um formato universal de dados e pode ser utilizado em vários meios, tais como páginas Web ou aplicativos *desktop*. Nesse ponto do artigo o Data Module deve se parecer com a **Figura 6**.

Vamos agora fazer a primeira parte da programação no sistema. Devemos nesse ponto, codificar dois eventos no Data Module, mais especificamente no *cdsFuncionarios*. Codificaremos os eventos *OnNewRecord* (ao incluir novo registro) e *OnCalcFields* (ao calcular campos).

Antes disso, devemos entender o que são eventos. Eventos são ações disparadas em determinados momentos e que devem executar alguma tarefa no aplicativo, como por exemplo: exibir uma mensagem, gravar um registro, efetuar um cálculo, fechar uma janela etc.

Os eventos *OnNewRecord* e *OnCalcFields* acontecem quando um novo registro é adicionado à tabela e quando um novo cálculo acontece em um *ClientDataSet*, respectivamente. Inicialmente codificaremos o evento *OnCalcFields* e para isso clique no *cdsFuncionarios*, selecione a janela de propriedades do Delphi no *Object Inspector*, clique na aba *Events* e por último clique duas vezes no evento *OnCalcFields*. O Delphi então criará o cabeçalho do evento como no exemplo a seguir:

```
procedure TdmodPrincipal.
  cdsFuncionariosCalcFields(DataSet: TDataSet);
begin
end;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Everson Volaco, mostrando como trabalhar com arquivos XML no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=1624

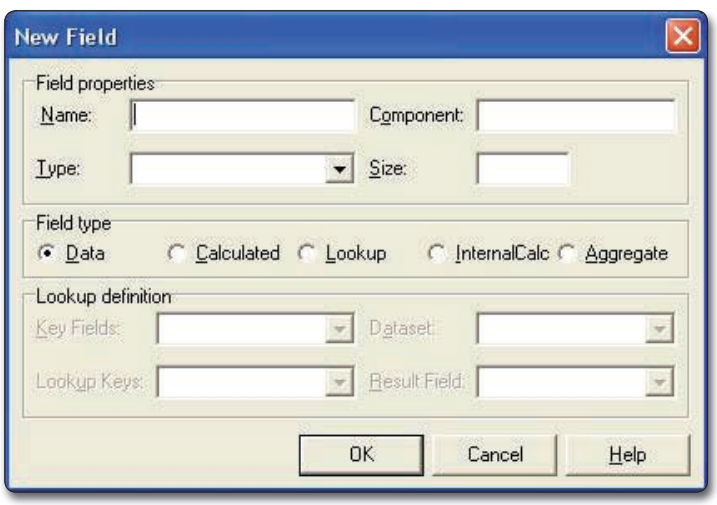


Figura 4. Diálogo de inclusão de campos

Campo	Tipo/Tamanho
ID_FUNCIONARIO	Autoinc
ID_DEPARTAMENTO	Integer
NOME	String(50)
ENDERECO	String(100)
CIDADE	String(30)
ESTADO	String(2)
CEP	String(9)
TELEFONE	String(15)
EMAIL	String(50)
FOTO	String(255)
VLR_HORA	Float
DIAS_TRABALHADOS	Integer

Tabela 1. Tabela Funcionários

Agora basta digitarmos o código correto que fará o cálculo. Digite o código a seguir:

```
cdsFuncionarios.FieldByName('SALARIO').
  AsFloat := (cdsFuncionarios.FieldByName(
  'VLR_HORA').AsInteger * 8 *
  cdsFuncionarios.FieldByName(
  'DIAS_TRABALHADOS').AsInteger);
```

Esse código atualiza o campo SALARIO com o resultado da multiplicação dos campos VLR_HORA e DIAS_TRABALHADOS. Com isso, toda vez que o campo VLR_HORA ou DIAS_TRABALHADOS sofrer uma alteração, o SALARIO será atualizado automaticamente.

Agora codificaremos o evento *OnNewRecord*. Novamente no *Object Inspector*, encontre o evento indicado e clique duas vezes nele. Por fim, digite o código a seguir:

```
cdsFuncionarios.FieldByName('CIDADE').
AsString := 'SÃO PAULO';
cdsFuncionarios.FieldByName('ESTADO').
AsString := 'SP';
```

Aqui estamos dizendo que o campo CIDADE inicialmente terá o conteúdo "SÃO PAULO" toda vez que um novo funcionário for adicionado. Do mesmo modo, o campo ESTADO terá o valor "SP". Isso facilita a vida do usuário do software já que poupa trabalho na digitação de novos funcionários.

Nesse caso, fixamos os valores supondo que os funcionários sejam de São Paulo-Capital, mas nada impede que o usuário altere para outro estado, ou seja, fica apenas como sugestão. Nossa última providência agora será vincular o *cdsFuncioarios* ao *cdsDeparamentos* de forma que ao clicar em um *Departamento* apenas os funcionários desse sejam exibidos na tela.

Criando índices e regras de validação

Clique uma vez no *cdsFuncionarios* e em seguida clique duas vezes na propriedade *IndexFields* no *Object Inspector*, pois vamos adicionar um índice à tabela. Na caixa de diálogo que se abre, clique com o botão direito e escolha *Add*. Criado o índice, selecione-o e no *Object Inspector*, na propriedade *Fields* digite: "ID_FUNCIONARIO;ID_DEPARTAMENTO".

Depois em *Options* marque as opções *ixPrimary* e *ixUnique*. Por fim, clique no *cdsFuncionarios* e configure as seguintes propriedades:

- *MasterFields*: campos que serão usados para vínculo, nesse caso digite apenas "ID";
- *IndexName*: índices que farão o controle de novos registros no banco e também é usado para vínculo. Selecione o único índice existente nessa tabela, criado anteriormente.

As duas e últimas grandes alterações que faremos neste exemplo serão feitas também no *cdsFuncionario*. Vamos incluir duas *Contraints* que são usadas para evitar que o usuário cadastre algo errado.

Clique no *cdsFuncionarios* e acesse a propriedade *Contraints*. Na caixa de diálogo que se abre, clique com o botão direito e escolha *Add*. Selecione então a *Constraint* criada e configure as suas propriedades: em *CustomConstraint* digite "VLR_HORA > 0" e em *ErrorMessage* digite "O campo Valor da Hora deve ser maior que zero".

Repita esse processo incluindo uma nova *constraint* com os dados a seguir:

- *CustomConstraint*: "DIAS_TRABALHADOS > 0 and DIAS_TRABALHADOS <= 20";
- *ErrorMessage*: "O campo Dias Trabalhados deve estar entre 1 e 20 dias".

Na primeira *constraint*, estamos dizendo que o VLR_HORA deve ser maior que zero, do contrário uma mensagem será exibida. Na segunda expressão, queremos que os dias trabalhados estejam entre 1 e 20.

As *constraints* são importantíssimas, pois evitam erros básicos de digitação, e como também servem como regras de validação do sistema. Pronto, agora a primeira etapa está concluída, vamos então criar a parte visual do sistema.

Desenhando a parte visual

Faremos uma tela simples para cadastrarmos os departamentos e funcionários. A tela sugerida pode ser vista na **Figura 7**.

Usaremos duas abordagens pra entendermos plenamente como funciona um sistema com banco de dados. Na parte superior (onde serão cadastrados os Departamentos) colocaremos um *DBNavigator*, um *Button* e um *DBGrid*. Na inferior (cadastro de funcionários) incluiremos botões personalizados para manipulação dos dados do sistema.

Para acesso ao banco de dados, incluiremos dois *Data Sources*. Esses componentes são usados para vincular nossos componentes visuais, que mostrarão os dados das tabelas, aos *ClientDataSets*.

> **Nota:** Antes de iniciarmos o desenvolvimento da janela principal clique em *File>Use unit*. Selecione *dmPrincipal* e clique em OK. Isso é necessário para que nossa tela tenha acesso aos componentes *cdsDepartamentos* e *cdsFuncionarios* no Data Module.

Inclua um *DBNavigator* da paleta *Data Controls* do Delphi e também um *Data Source* da paleta *Data Access*. Mude o *Name* do *DataSource* para "dtsDeparta-

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli, mostrando como trabalhar com constraints no Delphi.
www.devmedia.com.br/articles/viewcomp.asp?comp=567

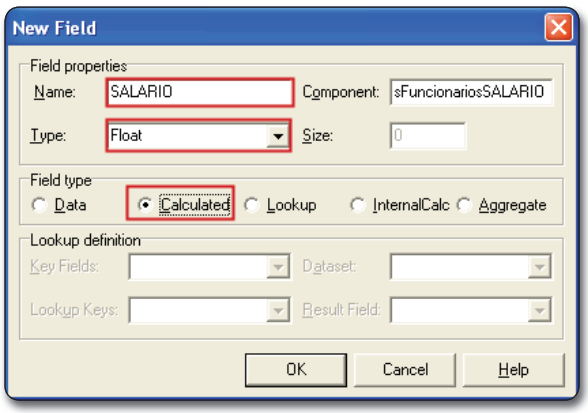


Figura 5. Criando o campo Salário

Figura 6. Data Module com os componentes criados

mentos". Logo após, modifique a propriedade *DataSet* do *dtsDepartamentos* apontando-o para o *cdsDepartamentos* no Data Module.

Clique no *DBNavigator* e em sua propriedade *DataSource* selecione o *dtsDepartamento*. Por último, insira um *DBGrid* da paleta *Data Controls* e mude sua propriedade *DataSource* escolhendo também o *dtsDepartamento*.

Para que apareçam os campos no *DBGrid*, basta clicar duas vezes no componente que uma pequena caixa de diálogo aparecerá. Nessa caixa clique em *Add all fields* e os campos ID e DEPARTAMENTO serão mostrados (**Figura 8**).

O que precisamos fazer agora é colocar um botão ao lado do *DBNavigator*. Clicando duas vezes no botão, vamos ao seu evento *OnClick*. Digite o código a seguir:

```
dmodPrincipal.cdsDepartamentos.SaveToFile(
  ExtractFilePath(Application.ExeName) +
  'Departamentos.xml');
```

Esse código fará com que os dados, ou seja, os departamentos inseridos ou alterados sejam guardados no *Departamentos.xml*. Isso também é chamado de persistência de dados. Feitas as devidas configurações já temos um cadastro de Departamentos pronto e funcionando.

O *DBNavigator* tem todas as funcionalidades prontas, ou seja, inclusão, alteração, exclusão etc. Não é necessário fazer alterações nesse ponto.

Cadastro de funcionários

No cadastro de funcionários faremos uso da segunda abordagem: Codificaremos nossas próprias operações e entenderemos melhor como funcionam as rotinas de inclusão, alteração, exclusão etc. Adicione cinco *BitBtns* da paleta *Additional*, onde cada um receberá os nomes de "spbNovo", "spbAlterar", "spbApagar", "spbGravar" e "spbCancelar" da esquerda pra direita. Se preferir, adicione imagens em cada botão para diferenciá-los um dos outros.

> **Nota:** Para inserir imagens nos botões, acesse a propriedade *Glyph* e selecione uma imagem qualquer no disco.

Logo abaixo, adicione um *Label* e um *Edit* da paleta *Standard*. No *Label* mude a propriedade *Caption* para "Localizar". Limpe a propriedade *Text* do *Edit* e mude seu *Name* para "edtLocFunc". Adicione um novo *DataSource* e mude seu *DataSet* para *cdsFuncionarios* e seu *Name* para "dtsFuncionarios". Coloque também um *DBGrid* e em sua propriedade *DataSource* para *dtsFuncionarios*.

Adicione todos os campos da tabela funcionários ao *DBGrid* usando a mesma técnica anterior. Se preferir apague alguns campos do *DBGrid*. Deixe apenas os campos mais importantes: ID_FUNCIONARIO, ID_DEPARTAMENTO e NOME.

Ao lado do *DBGrid* vamos inserir os componentes necessários para visualizar os dados do banco, chamados de componentes visuais. Vamos inserir um *DBEdit* para cada campo exceto para FOTO. O *DBEdit* fica na paleta *Data Controls*. Esses componentes possuem duas propriedades importantes:

- *DataSource* que deve ser ligado ao *dtsFuncionarios*;
- *DataField* : que deve ser ligado ao campo da tabela.

Então, adicione dez *DBEdits*, sendo um para cada campo da tabela *Funcionarios*. Como esses componentes não possuem um rótulo para que o usuário saiba a informação que deve ser cadastrada, inclua também um *Label* para cada *DBEdit* tomando o cuidado para trocar a propriedade *Caption* para o título adequado assim como na **Figura 7**.

Por último insira três novos componentes: um *Image* (*Additional*), um *BitBtn* (*Additional*) e um *OpenDialog* (*Dialogs*).

Codificando o cadastro de funcionários

Como podemos ver, não codificamos praticamente nada ao desenvolver a primeira parte da tela principal. Usamos apenas um *DBNavigator* que já faz toda a interação com o banco de dados através de seus botões. Com ele conseguimos incluir, alterar, excluir, gravar etc.

Basicamente o que faremos é chamar os métodos de inclusão, alteração, exclusão, gravação e cancelamento, presentes nos componentes de acesso a dados. Só para entendermos bem, os componentes *Clien-*

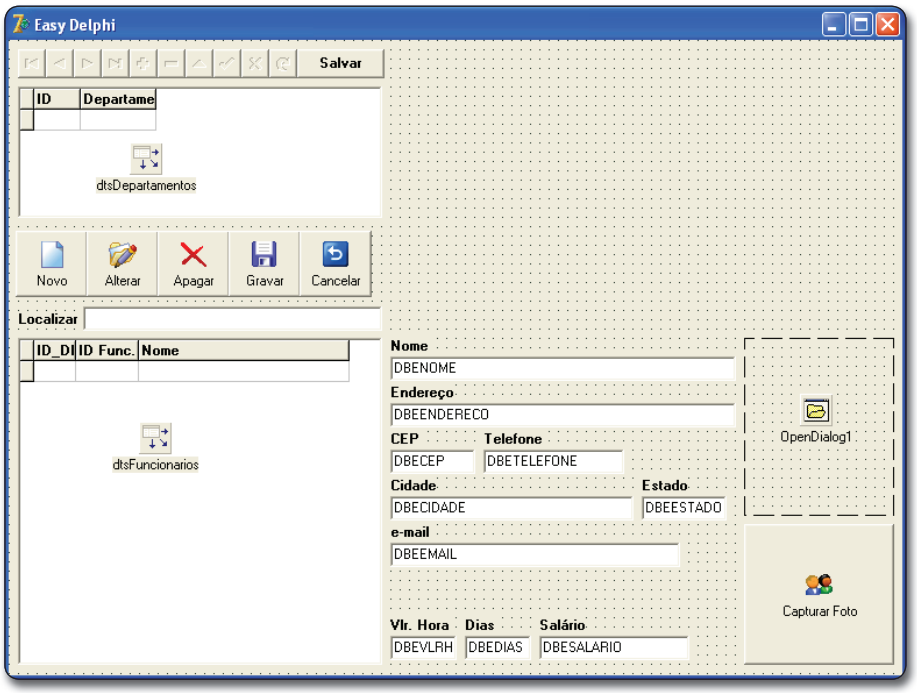


Figura 7. Sugestão de tela do sistema

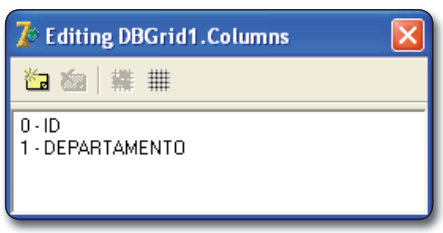


Figura 8. Campos no DBGrid

tDataSet que incluímos no Data Module possuem alguns métodos, que são rotinas prontas que executam uma determinada tarefa. As rotinas que usaremos são:

- *Append*: insere um novo registro no banco de dados (tabela);
- *Edit*: altera um registro na tabela;
- *Delete*: exclui um registro da tabela;
- *Post*: grava um registro que foi incluído ou alterado;
- *Cancel*: cancela a inclusão ou alteração.

Além de chamar os métodos necessários para as rotinas descritas anteriormente, vamos também ativar e desativar alguns botões na tela pra evitar que o usuário cometa erros. Começaremos pelo *Novo*. Clique duas vezes no *spbNovo* e inclua o código da **Listagem 1**.

No código anterior, verificaremos se o estado (*State*) da tabela é igual a *dsEdit* (edição) e caso seja, chamamos o *Exit* pra sair do evento sem nenhuma ação. Caso contrário, o *Append* do *ClientDataSet* é chamado.

Isso fará com que seja adicionado um novo funcionário na tabela. Nesse momento, todos os *DBEdits* vinculados ao *cdsFuncionarios* ficarão vazios esperando que sejam preenchidos com os dados do novo funcionário. Em seguida ativamos os botões *Gravar* e *Cancelar* e desativamos *Novo*, *Alterar* e *Excluir* pra evitar que o usuário clique novamente nos botões.

O código do *Alterar* é bastante semelhante ao *Novo*. Veja na **Listagem 2** que também verificamos se o registro não está sendo alterado e então chamamos o *Edit* ao invés de *Append*. Novamente ativamos e desativamos os botões conforme o necessário.

A **Listagem 3** mostra o código do *Gravar*. Nele, logo de início, verificamos se a tabela está vazia através do *IsEmpty* e se o estado dela é *dsBrowse*, ou seja, nem em edição, nem em inserção. Se esse for o caso, saímos do evento, senão gravamos o registro chamando o *Post* para gravação. Desativamos/ativamos botões e por último gravamos o arquivo *Funcionarios.xml*.

O *SaveToFile* manda as alterações e inclusões para o *Funcionarios.xml* e grava-as para posterior consulta. Se não fizéssemos isso, quando abríssemos novamente o sistema, nada estaria gravado. O código do *Cancelar* na **Listagem 4** é bastante semelhante, retirando apenas a chamada ao *SaveToFile*. Esse botão serve apenas para que possamos cancelar uma inclusão ou alteração.

Agora veremos o botão mais simples da aplicação, o *Excluir*. Nele, perguntamos ao usuário se realmente deseja apagar o registro atual e em caso positivo chamamos o *Delete* do *cdsFuncionario* e então salvamos o XML, conforme a **Listagem 5**.

Não há necessidade de ativar/desativar controles de tela. Outro botão interessante no sistema é o *Capturar Foto*. Nesse, capturamos uma foto do computador e gravamos o seu endereço no campo FOTO da tabela funcionários, como temos na **Listagem 6**.

Listagem 1. Código do Novo

```
procedure TfrmPrincipal.spbNovoClick(Sender: TObject);
begin
  if dmodPrincipal.cdsFuncionarios.State in [dsEdit] then
    Exit
  else
  begin
    dmodPrincipal.cdsFuncionarios.Append;
    spbCancelar.Enabled := True;
    spbGravar.Enabled := True;
    spbNovo.Enabled := False;
    spbExcluir.Enabled := False;
    spbAlterar.Enabled := False;
  end;
end;
```

Listagem 2. Código do Alterar

```
procedure TfrmPrincipal.spbAlterarClick(Sender: TObject);
begin
  if (dmodPrincipal.cdsFuncionarios.IsEmpty) or
    (dmodPrincipal.cdsFuncionarios.State in [dsInsert]) then
  exit
  else
  begin
    dmodPrincipal.cdsFuncionarios.Edit;
     spbCancelar.Enabled := True;
     spbGravar.Enabled := True;
     spbNovo.Enabled := False;
     spbExcluir.Enabled := False;
     spbAlterar.Enabled := False;
  end;
end;
```

Listagem 3. Código do Gravar

```
procedure TfrmPrincipal.spbGravarClick(Sender: TObject);
begin
  if (dmodPrincipal.cdsFuncionarios.IsEmpty) and
    (dmodPrincipal.cdsFuncionarios.State in [dsBrowse]) then
    Exit
  else
  begin
    dmodPrincipal.cdsFuncionarios.Post;
    spbCancelar.Enabled := False;
    spbGravar.Enabled := False;
    spbNovo.Enabled := True;
    spbExcluir.Enabled := True;
    spbAlterar.Enabled := True;
    dmodPrincipal.cdsFuncionarios.SaveToFile(
      ExtractFilePath(Application.ExeName) + 'Funcionarios.xml');
  end;
end;
```

Listagem 4. Código do Cancelar

```
procedure TfrmPrincipal.spbCancelarClick(Sender: TObject);
begin
  if (dmodPrincipal.cdsFuncionarios.IsEmpty) and
    (dmodPrincipal.cdsFuncionarios.State in [dsBrowse]) then
    Exit
  else
  begin
    dmodPrincipal.cdsFuncionarios.Cancel;
    spbCancelar.Enabled := False;
    spbGravar.Enabled := False;
    spbNovo.Enabled := True;
    spbExcluir.Enabled := True;
    spbAlterar.Enabled := True;
  end;
end;
```

A explicação é simples: em primeiro lugar estamos usando a palavra reservada *with* no início do código. Essa palavra é usada pra "economizarmos" digitação, ou seja, ao invés de digitarmos a todo instante a expressão:

```
dmodPrincipal.cdsFuncionarios.FieldByName(
  'FOTO').AsString
```

Digitamos diretamente *FieldByName ('FOTO').AsString*, pois as duas primeiras partes da expressão já estão declaradas no bloco *with..do*. Em seguida verificamos se o usuário selecionou um arquivo na caixa de diálogo e então atualizamos o campo da tabela (FOTO) e o componente de imagem (*imgFoto*).

Listagem 5. Código do Excluir

```
procedure TfrmPrincipal.spbExcluirClick(Sender: TObject);
begin
  if not (dmodPrincipal.cdsFuncionarios.IsEmpty) then
    if MessageDlg('Confirma exclusão desse '+
      'funcionário?',  mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
      dmodPrincipal.cdsFuncionarios.Delete;
      dmodPrincipal.cdsFuncionarios.SaveToFile(
        ExtractFilePath(Application.ExeName) + 'Funcionarios.xml');
    end;
end;
```

Listagem 6. Código do Capturar Foto

```
procedure TfrmPrincipal.BitBtn1Click(Sender: TObject);
begin
  with dmodPrincipal, cdsFuncionarios do
  begin
    if OpenDialog1.Execute then
    begin
      if not (State in [dsEdit, dsInsert]) then
        Edit;
      FieldByName('FOTO').AsString := OpenDialog1.FileName;
      imgFoto.Picture.LoadFromFile(FieldByName('FOTO').AsString);
    end;
  end;
end;
```

Listagem 7. Código o evento OnShow do form principal

```
procedure TfrmPrincipal.FormShow(Sender: TObject);
begin
  with dmodPrincipal do
  begin
      cdsDepartamentos.LoadFromFile(ExtractFilePath(Application.ExeName)+
          'Departamentos.xml'); cdsFuncionarios.LoadFromFile(
          ExtractFilePath(Application.ExeName)+'Funcionarios.xml');
    cdsFuncionarios.Open;
    cdsDepartamentos.Open;
  end;
end;
```

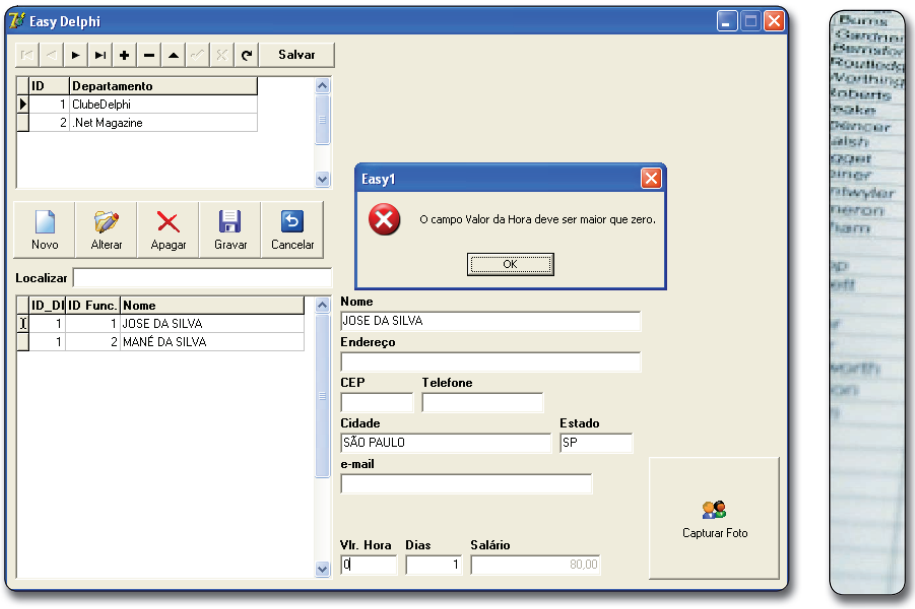


Figura 9. Sistema executando

Para finalizar, precisamos apenas criar uma pequena pesquisa onde poderemos digitar o nome do funcionário e procurar no cadastro. Por isso, incluímos o *edtLocFunc*. O *Edit* contém um evento chamado *OnChange*, que é disparado toda vez que temos uma alteração na propriedade *Text*, ou seja, toda vez que digitamos algo dentro dele.

Para fecharmos com chave de ouro nosso artigo, clique duas vezes no *edtLocFunc* e no evento *OnChange* digite o seguinte código:

```
dmodPrincipal.cdsFuncionarios.Locate('NOME',
edtLocFunc.Text, [loPartialKey,
loCaseInsensitive]);
```

Estamos usando outro método do *cdsFuncionarios*, o *Locate*, que recebe como parâmetro de entrada três opções:

- Campo onde será procurado o valor (*KeyFields*);
- Variável usada para pesquisa (*KeyValues*);
- Tipo de pesquisa, que pode ser (*Options*):
  - *LoPartialKey*: pesquisa parcial;
  - *LoCaseInsensitive*: pesquisa não sensível, ou seja, que não leva em consideração maiúsculas e minúsculas;

Veja na **Figura 9** o sistema em execução, com a validação do campo VLR_HORA.

Abrindo os DataSets

Para finalizar nosso exemplo precisamos codificar o evento *OnShow* do formulário principal. Nele iremos colocar uma chamada ao método *LoadFromFile* de cada *ClientDataSet* para que os dados contidos nos arquivos XML sejam exibidos e conseqüentemente fiquem disponíveis para alteração e/ou exclusão. Clique em uma área vazia de nosso formulário principal e vá até o evento *OnShow* no *Object Inspector*. Clique duas vezes e escreva o código da **Listagem 7**.

Conclusão

Neste artigo, criamos um pequeno sistema com acesso local à base de dados e entendemos melhor alguns dos principais conceitos de bancos de dados. Nas próximas edições veremos uma série de outras dicas de desenvolvimento para iniciantes. Um forte abraço e até a próxima ●

DevMedia
GROUP

O conteúdo ao seu alcance!

+ de **600 artigos** impressos por ano

+ de **1500 vídeo-aulas** de acesso imediato

+ de **24 cursos** online

+ de **130 vídeos inéditos** todos os meses

Se você busca **conteúdo** de qualidade e aprendizagem rápida.

Aqui é seu lugar!

Acesse e confira:

www.devmedia.com.br

Artigos para iniciantes na linguagem Delphi

SEÇÃO

Herança de formulários

Agilizando o desenvolvimento de telas de cadastros

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

No mundo relacional em que vivemos e aprendemos, fica difícil utilizar a Orientação a Objetos em nossos projetos. Talvez as próprias instituições de ensino tenham culpa, uma vez que ensinam os alunos a programar de forma estruturada e com banco de dados relacionais e, ao final do curso mostram pouco de como trabalhar com Orientação a Objetos.

Mas mesmo assim, podemos utilizar algumas técnicas mesmo que nosso sistema persista os dados em bancos relacionais. Neste artigo vamos aprender a utilizar a herança visual de formulários que ajudará a agilizar o desenvolvimento de telas de cadastros.

Caso de uso

Para aplicar o artigo na prática criaremos uma base de dados no Firebird, que terá uma tabela chamada CLIENTES e uma tabela para controlar o campo auto-incremento. Essa solução poderá ser utilizada em qualquer banco de dados, pois não utilizará *Trigger* e *Generators* específicos do Firebird, assim o sistema pode ser portado para outros bancos de dados. Na **Listagem 1** temos o *script* do banco que deverá ser criado.

> **Nota**: Para criar o banco de dados citado, você precisa ter o Firebird instalado (www.firebirdsql.org) e utilizar um utilitário como o IBExpert (www.ibexpert.com) para executar o script.

Criando a aplicação

Para esse artigo utilizarei o Delphi 7, mas fique a vontade para utilizar outra versão do Delphi que suporte aplicações Win32. Crie uma nova aplicação através do menu *File>New>Application*. Como o primeiro formulário é o principal (será executado ao iniciar o sistema) altere seu *Name* para "frmPrincipal" e salve o arquivo como "uPrincipal.pas". Salve também o projeto

informando "pHeranca.dpr" para o nome. Agora crie um Data Module através do menu *File>New>Data Module*. Altere sua propriedade *Name* para "dmDados" e salve o arquivo como "uDados.pas".

Para o acesso ao banco de dados usaremos um *SQLConnection* da paleta *dbExpress*. Altere seu *Name* para "Banco" e coloque *False* para a propriedade *LoginPrompt*. Clique duas vezes sobre o componente e configure a conexão informando o *Database* (caminho para o banco), bem como *User_Name* e *Password* caso sejam diferentes do padrão do Firebird.

> **Nota**: Não é o foco deste artigo demonstrar como funciona o dbExpress, consulte nosso portal para um curso completo sobre dbExpress (www.devmedia.com.br).

Não se esqueça de mudar o *SQLDialect* para "3", pois temos um campo do tipo *Date* que só é reconhecido nesse *Dialect*. Agora adicione e configure os seguintes componentes:

SQLDataSet1 (dbExpress)

- Mude a propriedade *SQLConnection* para *Banco*;
- *CommandText* para "select * from CLIENTES where CODCLIENTE = :pCODCLIENTE";
- *Name* para "sdsClientes";
- *Params* definir *ftInteger* para o parâmetro pCODCLIENTE;

Clique duas vezes sobre o componente para abrir o *Field Editor*. No editor, clique com o botão direito e escolha a opção *Add fields* para adicionar todos os campos. Selecione o CODCLIENTE, abra as sub-propriedade de *ProviderFlags* e coloque *True* para *pfInKey*. Selecione todos os campos restantes, abra as sub-propriedade de *ProviderFlags* e coloque *False* para *pfInWhere*.

> **Nota:** A configuração de *ProviderFlags* é muito importante pois determina como os campos serão adicionados nas instruções SQL de *insert*, *update* e *delete* que o *DasetProvider* montará automaticamente.

DataSetProvider1 (Data Access)

- Mude a propriedade *DataSet* para *sdsClientes*;
- *Name* para "dspClientes";
- *UpdateMode* para *upWhereKeyOnly*.

ClientDataSet1 (Data Access)

- Mude a propriedade *ProviderName* para *dspClientes*;
- *Name* para "cdsClientes";

Clique com o botão direito do mouse sobre o componente e escolha o item *Fetch Params* para buscar o parâmetro que existe no *SQLDataSet*.

SQLDataSet2 (dbExpress)

- Mude a propriedade *SQLConnection* para *Banco*;
- *CommandText* para "select * from CODIGOS where CAMPO = :pCAMPO";
- *Name* para "sdsCodigos";
- *Params* definir *ftString* para o parâmetro pCAMPO;

Clique duas vezes sobre o componente para abrir o *Field Editor*. No editor, clique com o botão direito e escolha a opção *Add fields* para adicionar todos os campos. Selecione o CAMPO, abra as sub-propriedade de *ProviderFlags* e coloque *True* para *pfInKey*. Selecione o VALOR, abra as sub-propriedade de *ProviderFlags* e coloque *False* para *pfInWhere*.

DataSetProvider2 (Data Access)

- Mude a propriedade *DataSet* para *sdsCodigos*;
- *Name* para "dspCodigos";
- *UpdateMode* para *upWhereKeyOnly*.

Caption	Name	Glyph	Hint	ShowHint
&Novo	btnNovo	FileNew	Inserir	True
&Alterar	btnAlterar	Edit	Editar	
&Gravar	btnGravar	FileSave	Salvar	
&Cancelar	btnCancelar	Undo	Desistir	
&Deletar	btnDeletar	Erase	Excluir	
&Fechar	btnFechar	DoorOpen	Sair	

Tabela 1. Configuração necessária nos Bitbtns

ClientDataSet2 (Data Access)

- Mude a propriedade *ProviderName* para *dspCodigos*;
- *Name* para "cdsCodigos";

Clique com o botão direito do mouse sobre o componente e escolha o item *Fetch Params* para buscar o parâmetro que existe no *SQLDataSet*. Veja na **Figura 1**, como ficou o Data Module da aplicação.

Figura 1. Data Module

Listagem 1. Script do banco para exemplo

```
CREATE TABLE CLIENTES (
  CODCLIENTE       INTEGER NOT NULL,
  NOME             VARCHAR(50) NOT NULL,
  DATANASCIMENTO   DATE NOT NULL,
  EMAIL            VARCHAR(200),
  SITE             VARCHAR(200),
 PRIMARY KEY ("CODCLIENTE"));

CREATE TABLE CODIGOS (
  CAMPO            VARCHAR(50) NOT NULL,
  VALOR            INTEGER NOT NULL,
 PRIMARY KEY ("CAMPO"));
```

Listagem 2. Função para campo auto-incremento

```
uses Variants;
...
function TdmDados.
  Auto(Campo: string): integer;
begin
  Result := 0;
  cdsCodigos.Close;
  cdsCodigos.Params.
    ParamValues['pCAMPO'] := Campo;
  cdsCodigos.Open;
  if ((cdsCodigos['VALOR'] > 0) and
    (cdsCodigos['VALOR'] <> null)) then
  begin
    cdsCodigos.Edit;
    cdsCodigos['VALOR'] :=
      cdsCodigos['VALOR'] + 1;
  end
  else
  begin
    cdsCodigos.Append;
    cdsCodigos['CAMPO'] := Campo;
    cdsCodigos['VALOR'] := 1;
  end;
  cdsCodigos.Post;
  Result := cdsCodigos['VALOR'];
  cdsCodigos.ApplyUpdates(0);
  cdsCodigos.Close;
end;
```

Listagem 3. Código dos botões da aplicação

```
uses uDados, DBClient;
...
Novo
procedure TfrmModelo.btnNovoClick(Sender: TObject);
begin
  if not dsPadrao.DataSet.Active then
     dsPadrao.DataSet.open;
  dsPadrao.DataSet.Append;
  (dsPadrao.DataSet as TClientDataset).FieldByName(
    dbChave.DataField).AsInteger := dmDados.Auto(dbChave.DataField);
  //Botões
  btnNovo.Enabled := False;
  btnAlterar.Enabled := False;
  btnGravar.Enabled := True;
  btnCancelar.Enabled := True;
  btnDeletar.Enabled := False;
  btnFechar.Enabled := False;
end;

Alterar
procedure TfrmModelo.btnAlterarClick(Sender: TObject);
begin
  dsPadrao.DataSet.Edit;
  //Botões
  btnNovo.Enabled := False;
  btnAlterar.Enabled := False;
  btnGravar.Enabled := True;
  btnCancelar.Enabled := True;
  btnDeletar.Enabled := False;
  btnFechar.Enabled := False;
end;

Gravar
procedure TfrmModelo.btnGravarClick(Sender: TObject);
var
  Errors: integer;
begin
  if Verifica_Campos_Em_Branco then
  begin
    dsPadrao.DataSet.Post;
    Errors := (dsPadrao.DataSet as TClientDataset).ApplyUpdates(0);
    if Errors > 0 then dsPadrao.DataSet.Edit
    else
    begin
      //Botões
      btnNovo.Enabled := True;
      btnAlterar.Enabled := True;
      btnGravar.Enabled := False;
      btnCancelar.Enabled := False;
      btnDeletar.Enabled := True;
      btnFechar.Enabled := True;
    end;
  end
  else
    Abort;
end;

Cancelar
procedure TfrmModelo.btnCancelarClick(Sender: TObject);
begin
  dsPadrao.DataSet.Cancel;
  (dsPadrao.DataSet as TClienTDataSet).CancelUpdates;
  //Botões
  btnNovo.Enabled := True;
  btnAlterar.Enabled := True;
  btnGravar.Enabled := False;
  btnCancelar.Enabled := False;
  btnDeletar.Enabled := True;
  btnFechar.Enabled := True;
end;

Deletar
procedure TfrmModelo.btnDeletarClick(Sender: TObject);
begin
  if MessageDlg('Deseja realmente excluir esse registro?',
    mtConfirmation, [mbYes,mbNo],0) = mrYes then
  begin
     dsPadrao.DataSet.Delete;
     (dsPadrao.DataSet as TClientDataSet).ApplyUpdates(0);
     dsPadrao.DataSet.Close;
     dsPadrao.DataSet.Open;
  end;
end;

Fechar
procedure TfrmModelo.btnFecharClick(Sender: TObject);
begin
 if dsPadrao.state in [dsEdit, dsInsert] then
   if MessageDlg('Deseja salvar alterações?',
     mtConfirmation,[mbok, mbcancel], 0) = mrOk then btnGravarClick(nil);
 Close;
end;
```

Função para campo auto-incremento

Como a idéia deste artigo é servir para qualquer banco de dados, será necessário criar uma função para buscar o campo auto-incremento que estará salvo na tabela denominada CODIGOS. No Data Module declare a seguinte função na seção *public*:

```
function Auto(Campo: string): integer;
```

Pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho da função e codifique-o conforme a **Listagem 2**.

O código da listagem anterior será chamado passando como parâmetro o nome do campo que deve ser auto-incrementado. Inicialmente é feita uma consulta para verificar se um registro com aquele campo existe cadastrado.

Se existir é realizada uma edição no registro somando mais um, senão criamos um novo registro para o campo, iniciando com o valor 1. Como utilizamos o valor *null* no teste precisamos adicionar *Variants* na seção *uses* da unit.

Criando o formulário modelo

Agora vamos à parte essencial do artigo. Crie um novo formulário através do menu *File>New>Form*. Altere a propriedade *Name* para "frmModelo" e salve a unit como "uModelo.pas". Coloque um *DataSource* da paleta *Data Access* nesse formulário e altere seu *Name* para "dsPadrao".

Não ligue a propriedade *DataSet* desse componente por enquanto, pois isso será feito nos formulários herdados. Adicione ao formulário um *Label* alterando o *Caption* para "Código" e um *DBEdit* alterando seu *Name* para "DBChave" e *DataSource* para *dsPadrao*.

Adicione seis *BitBtns* da paleta *Additional* ao formulário e configure suas propriedade de acordo com a **Tabela 1**. Veja na **Figura 2**, como fica o formulário com os componentes.

> **Nota:** As imagens dos botões encontram-se em *C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons*.

O formulário padrão está com uma interface bem simples. Você pode melhorar isso com alguns *Panels* ou *Bevel*. Deixe o

visual do formulário de modo que você ache atraente e completo. Agora codifique os botões conforme a **Listagem 3**.

Para você entender os códigos da listagem anterior, é importante lembrar que esse formulário será nosso modelo e, por isso os códigos fazem referência ao *dsPadrao* na propriedade *DataSet*. Então, no componente que estiver ligado o *DataSet* é que será chamado o *Open*, *Append*, *Post* etc.

O *dbChave* é um *DBEdit* que será ligado, através da propriedade *DataField*, ao campo chave da tabela, caso esse seja auto incremento. Se não existir auto incremento da tabela basta selecionar o *DBEdit* e o *Label* para alterar a propriedade *Visible* para *False*, deixando-os escondidos.

No código do *Gravar* chamamos *Verifica_Campos_Em_Branco* que dever ser declarada na seção *private* da unit, da seguinte forma:

```
function Verifica_Campos_Em_Branco: Boolean;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Éverson Volaco, mostrando como criar formulários com herança visual.

www.devmedia.com.br/articles/viewcomp.asp?comp=1716

Listagem 4. Código da função Verifica_Campos_Em_Branco

```
function TfrmModelo.Verifica_Campos_Em_Branco: Boolean;
var
  cont: integer;
begin
  Result := True;
  for cont := 0 to dsPadrao.DataSet.FieldCount - 1 do
    if dsPadrao.DataSet.Fields[cont].Required then
      if (dsPadrao.DataSet.Fields[cont].IsNull) or (dsPadrao.DataSet.Fields[
        cont].AsString = '') then
      begin
        MessageDlg('Preencha o campo "'+ dsPadrao.DataSet.Fields[
          cont].DisplayLabel + '"', mtWarning, [mbOk], 0);
        Result := False;
        dsPadrao.DataSet.Fields[cont].FocusControl;
        Abort;
      end;
end;
```

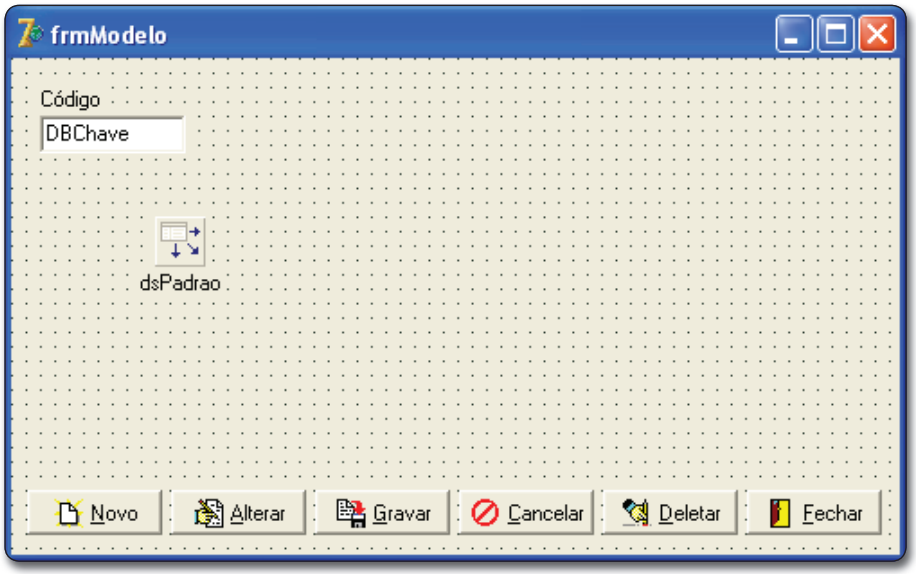


Figura 2. Formulário padrão

“ O Maior Portal para Desenvolvedores da América Latina.”

Acesse:

www.devmedia.com.br

DevMedia GROUP

Pressione CTRL+SHIFT+C e codifique a função conforme é exibido na **Listagem 4**.

No evento *OnClose* do formulário devemos fechar o *DataSet* com o seguinte código:

```
dsPadrao.DataSet.Close;
```

Para finalizar o formulário modelo é necessário adicionar na seção *uses* da unit, o *DBClient* para que seja reconhecido o tipo *TClientDataSet* e a unit do Data Module.

Usando o formulário modelo

Agora chegou a hora de utilizar o formulário modelo. Devemos criar um novo formulário, mas não pela opção *File>New>Form*. Acesse o menu *File>New>Other* e note que existe uma aba com o nome do projeto (*pHeranca*). Acesse a aba e selecione o *frmModelo*. Na parte inferior que existe uma opção marcada como herança (*Inherit*), como vemos na **Figura 3**.

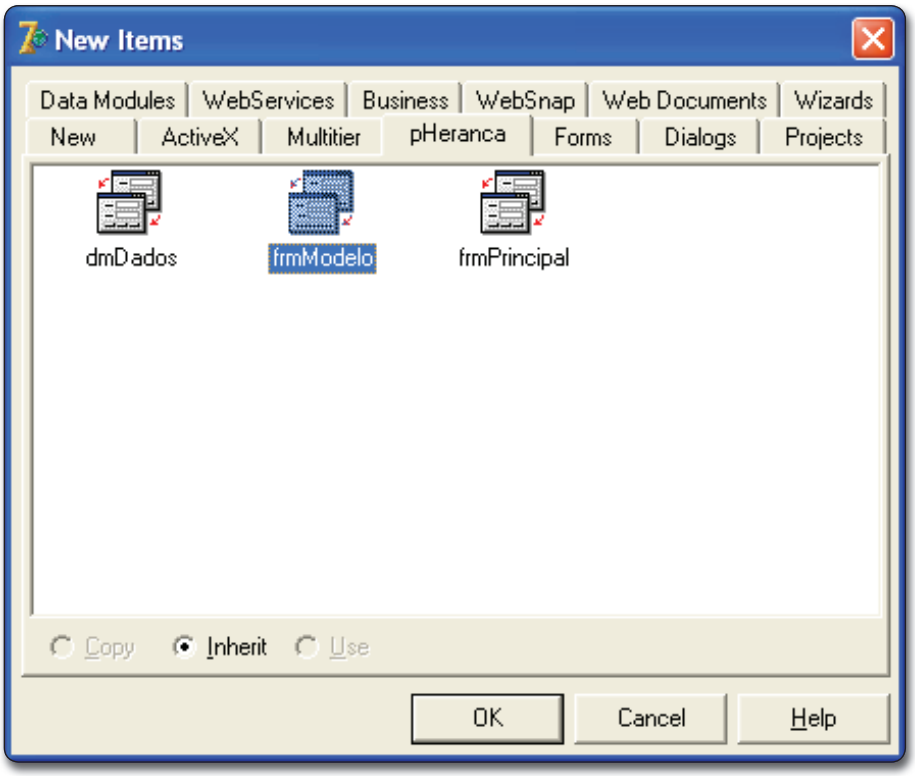


Figura 3. Herdando um formulário

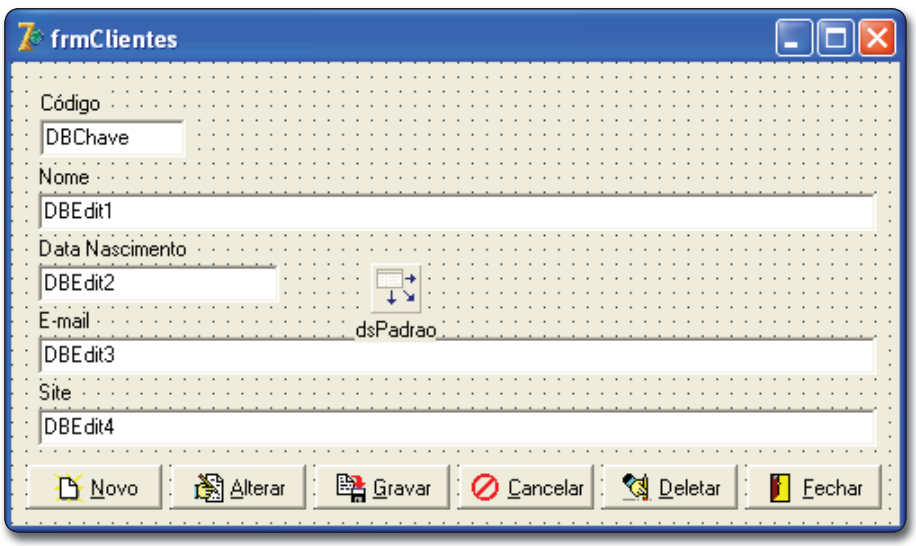


Figura 4. Formulário herdado

Clique em OK. Altere a propriedade *Name* para "frmClientes" e salve a unit como "uClientes.pas". Acesse o menu *File>Use Unit* e escolha *uDados*. Selecione o *DataSource* e aponte a propriedade *DataSet* para *dmDados.cdsClientes*.

Selecione o *dbChave* e altere a propriedade *DataField* para CODCLIENTE. Agora coloque um *Label* e *DBEdit* para cada campo da tabela de Clientes, fazendo as devidas ligações (**Figura 4**).

Pronto, agora é só criar um menu para chamar o formulário de Clientes e testar sua funcionalidade. Veja que para adicionar mais formulários de manutenção de dados ficou muito simples, pois toda a codificação está em um único local, o que facilita também para acertos e manutenção, tanto de código como de *layout*. Basta adicionar um novo componente no formulário modelo para ter o mesmo em todos os formulários herdados.

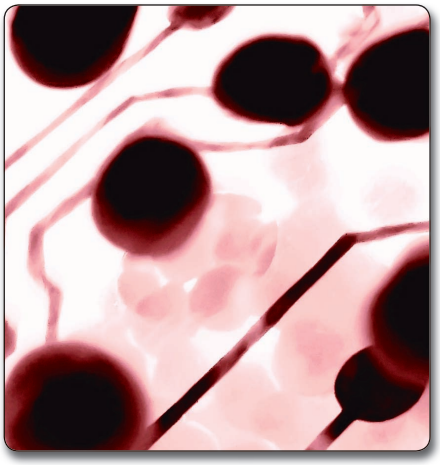
Conclusão

A herança é uma ótima opção para facilitar nossas vidas. Cada vez mais devemos pensar em interfaces genéricas que sirvam para várias interfaces diferentes e até projetos diferentes. Assim sua equipe ganhará em produtividade. Comece a utilizar OO em seus projetos através da herança. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista ao curso online de Luciano Pimenta, que mostra técnicas de herança visual em formulários e relatórios e mais dicas de POO.

www.devmedia.com.br/articles/viewcomp.asp?comp=4080



A EDIÇÃO QUE VOCÊ PRECISA ESTÁ ESGOTADA?

SEUS PROBLEMAS ACABARAM!!!

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

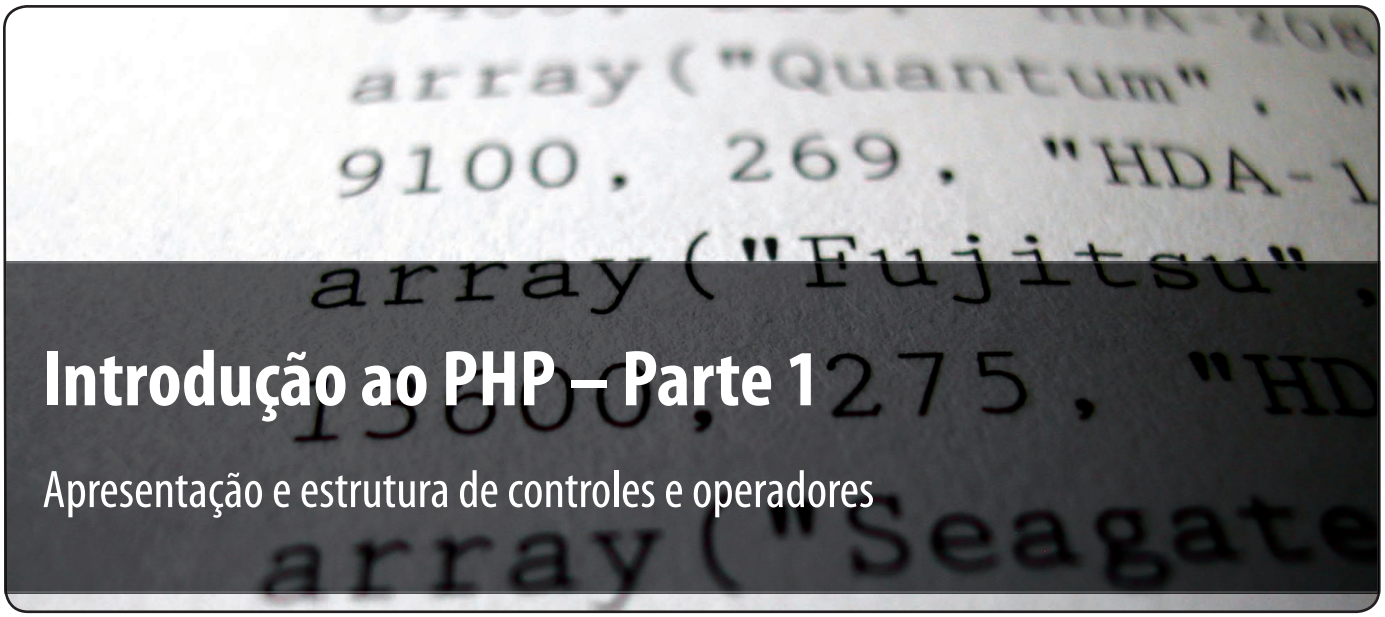
Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine.

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP



Introdução ao PHP – Parte 1

Apresentação e estrutura de controles e operadores

Ewald Geschwinde
Hans-Juergen Schoenig

Neste artigo, aprenderemos a escrever aplicações PHP simples, onde mostraremos os fundamentos do PHP. A primeira aplicação que encontraremos na maior parte dos livros sobre programação de computadores, é a "Hello World" (*Olá Mundo*). *Hello World* é fácil de implementar em todas as linguagens, pois não é preciso fazer nada além da exibição de uma seqüência na tela, portanto esse é um bom ponto de partida. O código a seguir, é a versão mais simples do *Hello World* em PHP:

```
<?php
  echo 'hello world<br>';
?>
```

Com apenas três linhas de código, podemos exibir uma mensagem simples na tela. A primeira linha de código, diz ao servidor Web que o código a seguir, tem que ser executado pelo interpretador do PHP.

Os caracteres *?>* indicam o fim do código. Utilizar a seqüência *<?php* não é o único modo possível de indicar ao servidor Web que o código PHP tem que ser executado. O PHP fornece três modos adicionais, como podemos ver na **Listagem 1**.

> **Nota:** Os exemplos mostrados neste artigo podem ser facilmente adaptados ao Delphi for PHP, basta usar o código dentro do bloco *<?php...?>* que aparece nos eventos da aba *JavaScript* do *Object Inspector*. Para outros eventos, basta usar o código normal, pois não temos o bloco *<?php...?>*.

Nas primeiras três linhas, podemos ver que não será necessário indicar que o interpretador do PHP tem que ser chamado. Esse método somente é possível quando as *short tags* estão habilitadas (podemos fazer isso configurando a opção *short open tag* no arquivo de configuração do PHP). No entanto, para deixar bem claro sobre a linguagem de programação que

estamos falando, recomendamos utilizar *<?php* em lugar de *<?*.

O terceiro método pode ser utilizado se as tags *Active Server Pages* (ASP) estiverem habilitadas. Se as ASP tags estiverem desligadas, *<%* e *%>* serão tratados como texto normal, e o código PHP não será interpretado. Portanto, recomendamos utilizar ou *<?php* ou *<script language="php">* para indicar ao servidor Web o que fazer com o código PHP. Neste artigo usaremos *<?php*.

Se tivermos colocado o arquivo que contém o código no diretório correto (ver a documentação e a configuração do servidor Web, para encontrar o diretório correto), podemos executar o *script* acessando-o no navegador Web. Se o arquivo for chamado *hello.php*, e se estiver no diretório HTML *default* da máquina local, pode ser facilmente acessado utilizando o endereço *http://localhost/hello.php*. Não é necessário acrescentar direitos de execução ao arquivo, porque o servidor Web só irá acessá-lo no modo leitura.

> **Nota:** Se estiver usando o IIS, basta copiar o arquivo para a pasta *C:\Inetpub\wwwroot*.

Listagem 1. Executando código PHP

```
<?
  echo 'hello world 1<br>';
?>

<script language="php">
  echo 'hello world 2<br>';

</script>
<%
  echo 'hello world 3<br>';
%>
```

Listagem 2. Segunda versão do Hello World

```
<html>
<head>
    <title>Title of the site</title>
</head>
<body>
    <?php echo 'hello world<br>'; ?>
</body>
</html>
```

Listagem 3. Gerando código HTML dinamicamente

```
<?php
  echo '<html>
    <head>
    <title>Title of the site</title> '.
    '</head>';
  echo "<body>hello world<br></body></html>\n";
?>
```

Listagem 4. Exemplo do uso de variáveis

```
<?php
  $text="hello world";
 echo '<html><head><title>Title of the site</title>'.
    '</head><body>';
  echo "$text<br>";
  echo "$text<br>";
  echo '</body></html>';
?>
```

Se tudo tiver sido feito corretamente, o PHP terá sido configurado com sucesso, e o servidor Web terá sido iniciado, assim, *Hello World* será exibido na tela. A primeira versão do *Hello World* produz apenas texto. No entanto, o objetivo principal de um programa PHP é gerar código HTML. No exemplo da **Listagem 2**, veremos um pequeno *script* para gerar a versão HTML de *Hello World*.

Como podemos ver, o código PHP está incluído no código HTML. A seqüência *<?php* indica ao servidor que o interpretador PHP deve ser iniciado. No exemplo, podemos ver que algumas partes do documento serão partes estáticas e outras serão criadas dinamicamente pelo PHP.

Outro modo de atingirmos o mesmo objetivo seria gerar o documento inteiro dinamicamente. Isso seria mais lento do que a versão que acabamos de ver, mas em alguns casos faz sentido ou seria até necessário gerar tudo dinamicamente, como temos o código na **Listagem 3**.

O primeiro comando *echo* compõe-se de duas partes. As três primeiras linhas estão entre aspas simples. O PHP reconhece que o comando continua na linha seguinte e que não acaba na primeira linha do *echo*. Dessa maneira, linhas múltiplas são tratadas como um único comando.

Outro modo de fazermos isso, seria utilizar uma operação *string*, como podemos ver ao final da linha de número três do *echo*. Depois das aspas simples, usamos um ponto para indicar ao PHP que a primeira seqüência, que acaba de terminar, deve ser concatenada com a seqüência seguinte. Nesse caso, a segunda seqüência é *</head>*. Todos os componentes do primeiro *echo* serão passados ao PHP entre aspas simples.

O segundo *echo* contém o texto que queremos que seja exibido na tela. Dessa vez, passamos o texto a ser exibido utilizando aspas duplas. A utilização de aspas simples ou aspas duplas faz uma diferença significativa: aspas simples são utilizadas para passar texto estático para o PHP. A utilização de aspas duplas permite que utilizemos variáveis dentro da seqüência.

O conhecimento dessa diferença é importante, pois de outra forma, poderíamos ter problemas. Executando o *script* que acabamos de ver, o resultado será um documento HTML simples:

```
<html>
  <head><title>Title of the site</title></
head>
  <body>hello world<br></body>
</html>
```

Esse código HTML ainda não será mais do que uma simples aplicação *Hello World*, mas já é código HTML.

Trabalhando com variáveis

Variável é um componente básico de quase toda linguagem de programação existente, e sem variáveis seria impossível escrever aplicações úteis. A idéia por trás do conceito de variáveis é a de ter um componente de linguagem que poderá conter valores diferentes.

É também possível utilizar variáveis como containeres para armazenar dados que serão utilizados em muitos lugares diferentes da aplicação. Imaginemos

um exemplo onde desejamos exibir duas vezes o mesmo texto na tela, utilizamos o código da **Listagem 4**.

Em primeiro lugar, atribuímos a variável chamada *$text*, a seqüência que desejamos exibir na tela. A seguir, o código HTML é gerado e a variável é exibida na tela duas vezes. Repare que temos que utilizar aspas duplas em vez de aspas simples, caso contrário, o resultado seria a exibição do nome da variável: *$text*. Com aspas simples, as variáveis não são utilizadas dentro de uma seqüência, portanto o resultado não seria o desejado. A **Figura 1** mostra o que é exibido.

Acrescentando comentários

A documentação é quase tão importante quanto o código de um programa. Atualmente, só 10% de todos os custos de produção de um software, são utilizados para escrever o código-fonte. Os 90% restantes, serão utilizados na manutenção do produto (informação foi fornecida pela Universidade de Viena).

Para reduzir os custos de manutenção de um produto e reduzir o tempo necessário para nos familiarizarmos com o código-fonte, é essencial acrescentar comentários aos programas. O PHP disponibiliza algumas formas simples de acrescentar comentários a um programa, como podemos ver no seguinte código:

```
<?php
    //Isso é um comentário
    /* Comentário estilo C */
    echo 'hello world';
?>
```

Em PHP os comentários são acrescentados ao código como em C e C++. O // diz ao interpretador que a linha inteira não deve ser executada. O /* ... */ é o antigo estilo C, que também é suportado pelo PHP. Os comentários no estilo C são utilizados para inserir comandos consistindo de múltiplas linhas:

```
<?php
  /* comentários estilo C
     maior que uma linha. */
?>
```

Recomendamos fazer uso intensivo de comentários para tornar o código mais claro e mais fácil de entender. Principalmente, se não formos os únicos que trabalharão com o código, os comentários facilitarão o dia-a-dia.

Estruturas de controle e operadores

Normalmente os programas precisam das chamadas estruturas de controle, que serão utilizadas para decidir em tempo de execução, o que o programa deve fazer. O controle de fluxo é uma parte importante de toda linguagem de programação e é essencial para construir aplicações. O PHP fornece uma larga variedade de métodos, como veremos.

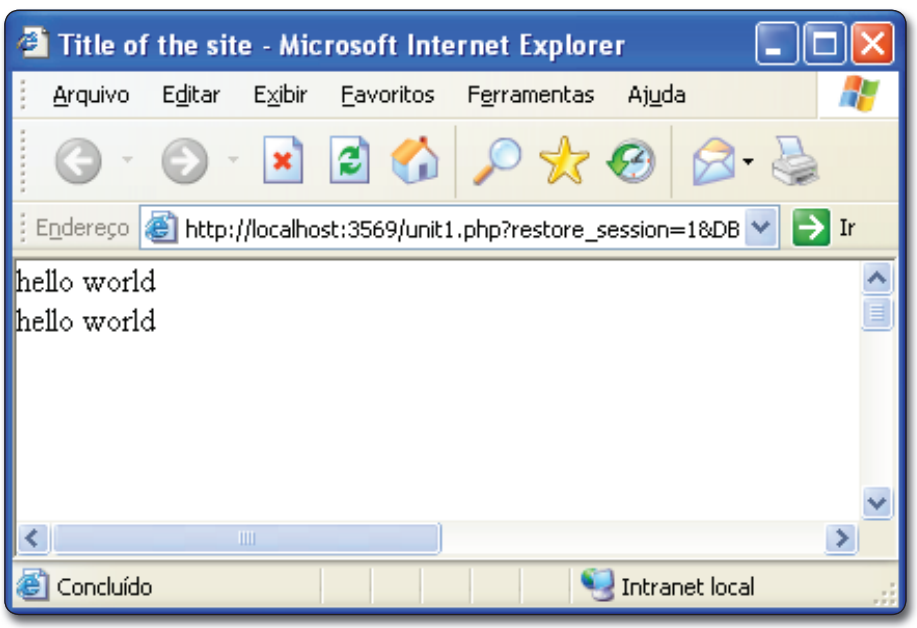


Figura 1. Uma versão mais sofisticada do Hello World

If/Else/Elself

Utilizamos o *if* para verificar se certa expressão é verdadeira ou não. O *if* é suportado por todas as linguagens de programação comumente utilizadas, tais como C/C++, Delphi, Java, Perl e Phyton e, por isso, é um instrumento fundamental de todo programador. Para mostrar como as condições são verificadas utilizando o *if*, incluímos um pequeno exemplo:

```
<?php
  if (2 > 1) { echo '2 é maior do que 1<br>';
}
?>
```

Uma mensagem será exibida na tela, porque 2 é maior do que 1. Se a condição não fosse cumprida, nada seria exibido.

> **Nota:** Se o PHP retornar um documento vazio, alguns navegadores levantarão um erro.

Normalmente, o *if* é utilizado para decidir se algo deve ser feito. Se certa condição for cumprida, o PHP executará o código entre colchetes, mas o que deve ser feito se uma condição não for cumprida? O *if* muitas vezes é utilizado em combinação com o *else*.

Sempre que uma expressão no *if* retorna *false*, o código sob o *else* será executado. A utilização do *if* em combinação com o *else*, é uma característica extremamente interessante, pois torna a programação muito mais fácil e ajuda a evitar erros de codificação. No exemplo da **Listagem 5** aprendemos a utilizar o *if* e *else*.

Em primeiro lugar inicializamos duas variáveis. Para *$a* atribuímos o valor 23 e à *$b* atribuímos o valor 34. A seguir, verificamos se *$a* é maior do que *$b*. Se a expressão retornar *true*, o texto "a é maior que b" será exibido na tela. Caso contrário, o texto "b é maior que a" será o resultado do programa.

No exemplo da **Listagem 6**, veremos o que podemos fazer, se quisermos que uma mensagem seja exibida caso *$b* seja maior do que *$a* e, ao mesmo tempo, *$b* seja maior do que 30.

A variável *$a* é menor do que *$b*, portanto, o PHP executará o código do *else*. No *else*, utilizamos um novo *if* para descobrir se *$b* é maior do que 30. É fácil resolver o problema da maneira como foi mostrado

no exemplo anterior, mas não é a forma mais elegante, assim poderíamos utilizar o código da **Listagem 7**.

Em vez de utilizar um segundo *if* no código do *else*, podemos também utilizar o *elseif*, que na maior parte dos casos é a forma mais enxuta e elegante. Às vezes, é necessário descobrir se duas ou mais condições serão cumpridas.

Para tal, expressões múltiplas são combinadas utilizando operadores e parênteses. Supondo que desejamos verificar se *$a* é maior do que 10, porém menor do que 50 usamos o código da **Listagem 8**.

Podemos utilizar o && para indicar ao PHP que ambas as condições têm de ser cumpridas: caso contrário, será retornado *false* e o código do *else*, que não existe no exemplo, seria executado. A expressão && é equivalente a *and*, portanto, não importa se usarmos *$a> 10 and $a <50* ou *$a> 10 && $a <50* no exemplo.

A utilização do && em vez de *and* é a forma mais comum, já que && também é suportado por linguagens de programação tais como o C/C++ e o Perl. Se executarmos o exemplo, o resultado será "23 é maior que 10 e menor que 50". A contrapartida para o *and* é o *or*. Se o *or* for utilizado, será retornado *true* quando ao menos uma das expressões retorna *true*. Vejamos um exemplo na **Listagem 9**.

Queremos que uma seqüência seja exibida na tela, caso *$a* for maior do que 10 ou menor do que 50. Se uma dessas condições for cumprida, a seqüência será exibida. O número 2 não é maior do que 10, mas é menor do que 50, portanto "2 é maior que 10 ou menor que 50" poderia ser exibido na tela. Em vez de || podemos utilizar também *or*.

Outro operador importante é o *xor*. Será retornado *false* se ambos os valores forem idênticos, caso contrário, o resultado é *true*. O exemplo da **Listagem 10** demonstra o uso do *xor*.

Como seria de se esperar, o *script* retorna "18 e 19 não são iguais".

Até agora, aprendemos a utilizar &&, || e *xor*. Agora está na hora de utilizar esses operadores como uma equipe para construir expressões mais complexas. Suponhamos que desejamos que uma seqüência seja exibida quando *$a* for maior do que 10 e menor do que 20 ou quando for exatamente igual a 2. Nesse

Listagem 5. Utilizando If e Else

```
<?php
  $a=23;
  $b=34;
  if ($a > $b)
  {
    echo 'a é maior que b';
  }
  else
  {
    echo 'b é maior que a';
  }
?>
```

Listagem 6. Utilizando If após o Else

```
<?php
  $a=23;
  $b=34;
  if ($a > $b)
  {
    echo 'a é maior que b';
  }
  else
  {
    if   ($b > 30)
    {
      echo 'b é maior que a e maior que 30.';
    }
  }
?>
```

Listagem 7. Utilizando ElseIf

```
<?php
  $a=23;
  $b=34;
  if ($a > $b)
  {
    echo 'a é maior que b';
  }
  elseif ($b > 30)
  {
    echo 'b é maior que a e maior que 30.';
  }
?>
```

Listagem 8. Utilizando múltiplas expressões

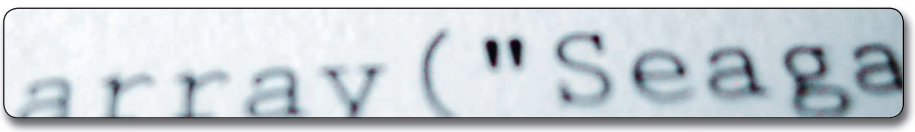
```
<?php
  $a=23;
  if ($a > 10 && $a < 50)
  {
    echo "$a maior que 10 e menor que 50";
  }
?>
```

Listagem 9. Utilizando o operador OR

```
<?php
  $a=2;
  if ($a > 10 || $a < 50)
  {
    echo "$a é maior que 10 ou menor que 50.";
  }
?>
```

Listagem 10. Utilizando o operador XOR

```
<?php
  $a=18;
  $b=19;
  if ($a xor $b)
  {
    echo "$a e $b são iguais";
  }
  else
  {
    echo "$a e $b não são iguais";
  }
?>
```



Listagem 11. Utilizando Group Expressions

```
<?php
  $a=2;
  if (($a > 10 && $a < 20) || $a == 2)
  {
    echo "$a maior que 10 mas menor que 20 ".
      "ou igual a 2";
  }
?>
```

Listagem 12. Exemplo simples de comparação

```
<?php
  $a=18;
  if ($a = $b)
  {
    echo '$b pode ser atribuído a $a';
  }
  else
  {
    echo '$b não pode ser atribuído a $a';
  }
?>
```

Listagem 13. Verificando se uma variável esta em uso

```
<?php
  if ($a)
  {
    echo '$a já foi definida';
  }
  else
  {
    echo '$a não foi definida ainda';
  }
?>
```

Listagem 14. Exemplo de While

```
<?php
  echo '<html><body>';
  $a=1;
  while ($a <= 5)
  {
    echo "a: $a<br>\n";
    $a++;
  }
  echo '</body></html>';
?>
```

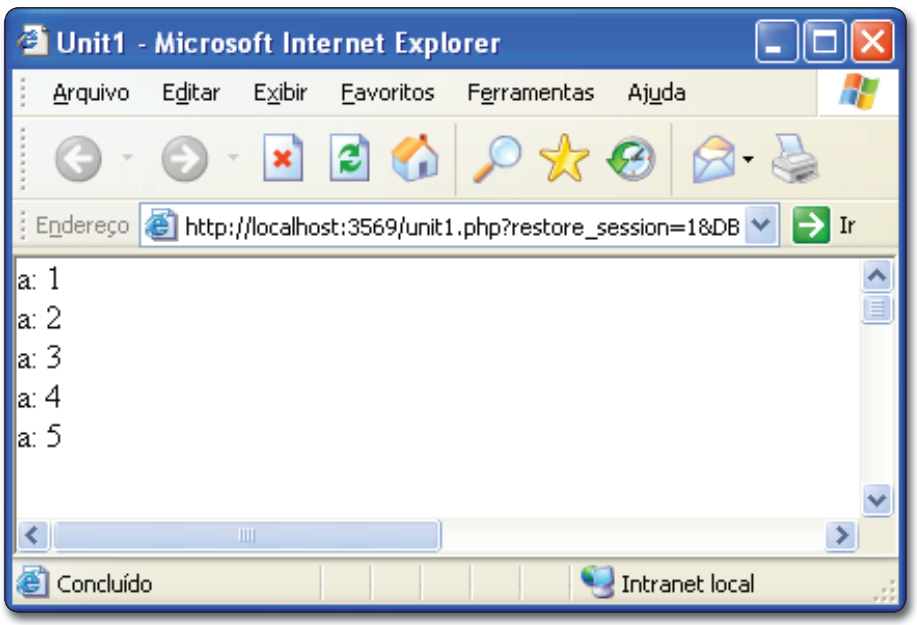


Figura 2. Usando um while

caso, serão necessárias *Group Expressions* como no exemplo da **Listagem 11**.

A primeira expressão compõe-se de duas partes, que serão conectadas utilizando o &&. Se todas as condições das expressões entre parênteses forem cumpridas, a primeira parte retornará *true*. Se tanto a primeira quanto a segunda parte da expressão inteira retornar *true*, o resultado será *true*. No caso, a segunda parte da expressão é um casamento exato, portanto a expressão inteira será verdadeira.

Aqueles que não têm ainda muita experiência com programação de computadores, podem achar estranho quanto ao por que de utilizarmos "= =" em vez de "=" no exemplo que acabamos de ver. Não se trata de um erro de impressão. O "= =" é utilizado para verificar se duas variáveis tem o mesmo conteúdo. Enquanto que "=" é utilizado para descobrir se uma variável pode ser atribuída a outra variável. Vejamos na **Listagem 12** um exemplo muito simples.

Em primeiro lugar, um valor é atribuído a *$a*. A seguir tentamos descobrir se *$b* pode ser atribuído a *$a*. Para isso, devemos utilizar o "=". Como *$b* não foi utilizado ainda, o código do *else* será executado. Os *ifs* podem também ser utilizados para descobrir se uma variável já foi definida. O exemplo da **Listagem 13** mostra como podemos descobrir se uma variável já esteve em uso.

Nesse caso o resultado será "$a não foi definida ainda", pois *$a* não foi utilizado anteriormente.

While do ... While

Repetições *while* são muito comuns em linguagens de programação. Como quase todas as demais linguagens de programação, o PHP provê repetições *while*, mas para que servem repetições *while*? São utilizadas para executar repetidas vezes um trecho do código. Vejamos no código da **Listagem 14** um exemplo de uma simples repetição *while*.

Os números de um até cinco serão exibidos. Em primeiro lugar, o código HTML será gerado. A seguir, atribuímos 1 a *$a* e iniciamos a repetição. O código entre as chaves ({}) será executado até que *$a* seja maior do que 5. Sempre que a repetição é processada, o literal *a:* e o conteúdo de *$a*

serão exibidos na tela, e a seguir *$a* será incrementado de 1.

O operador "++" é freqüentemente utilizado em linguagens como C e C++. Como podemos ver, está também disponível em PHP. A expressão *$a++* é equivalente a *$a=$a+1*. Veja na **Figura 2** o que será exibido pelo navegador quando executarmos o *script*.

O código HTML que foi gerado não é exibido na tela porque é utilizado só pelo navegador. A sintaxe anteriormente descrita representa a forma mais comum utilizada com repetições *while*. Contudo, o PHP fornece um modo adicional de utilização do *while*, como podemos ver na **Listagem 15**.

O código mostrado na listagem anterior é igual ao código que vimos anteriormente. A única diferença é a forma como o *while* foi utilizado. No segundo exemplo não utilizamos chaves, portanto, devemos sinalizar para o PHP quando a repetição termina, utilizando o *endwhile*.

Se estivermos utilizando repetições, devemos garantir que a condição que provoca a parada da repetição será falsa, depois de executar a repetição a quantidade desejada "n" vezes, caso contrário, a repetição ocorrerá até que um dos componentes (o PHP, o núcleo, o servidor Web e assim por diante) pare a execução do *script*.

No caso, teríamos uma repetição infinita caso *$a* nunca fosse incrementada. As repetições infinitas são perigosas e devemos lembrar disso. O PHP oferece uma opção no arquivo *php.ini*, para limitar tanto o consumo de memória quanto o tempo de execução de um *script* PHP. Recomendamos veementemente configurar essas opções com um valor razoável.

Às vezes, é necessário executar o código da repetição uma vez e, em seguida, verificar a condição para parar a repetição. Para conseguirmos isso, o PHP oferece as repetições *do...while*, como vemos na **Listagem 16**.

Em primeiro lugar, o valor 6 é atribuído a *$a* e, em seguida, iniciamos a utilização da repetição *do*. O valor *$a*, que é 6, é exibido na tela e logo em seguida verificamos se a repetição deve ser executada novamente. Como 6 é maior do que 5, o *echo* não será executado novamente. Nesse caso tivemos sorte, porque se *$a* fosse menor do que 6 teríamos uma repetição infinita. A variável *$a* não seria incrementada e, portanto, *$a <= 5* sempre seria *true*.

O que poderíamos fazer se tivesse que encerrar uma repetição no meio de um bloco? O PHP tem a solução adequada. O *break* pode ser utilizado para parar todos os tipos de repetições no meio de um bloco. Vejamos um exemplo na **Listagem 17**.

Queremos executar a repetição enquanto *$a* é menor do que 6, mas se *$a* for igual a 3, desejamos interromper a repetição *while*. Se executarmos o *script*, poderemos ver que serão exibidas duas linhas, pois a repetição foi interrompida após a exibição de duas linhas na tela.

Outra palavra-chave importante para trabalharmos com repetições é o *continue*. Ele é utilizado para parar a execução de um bloco sem interromper a repetição. A idéia é ignorar determinados passos da repetição. Vejamos um exemplo na **Listagem 18**.

A repetição é executada até que *$counter* seja igual a *$test*. Na repetição tentamos descobrir se *$counter* pode ser dividido por 2. Se o resultado for diferente de 0 (no caso, se for 1), incrementamos o contador e usamos *continue*. O programa come-

Listagem 15. Outro exemplo de While

```
<?php
  echo '<html><body>';
  $a=1;
  while ($a <= 5):
    echo "a: $a<br>\n";
    $a++;
  endwhile;
  echo '</body></html>';
?>
```

Listagem 16. Usando o do...while

```
<?php
  echo '<html><body>';
  $a=6;
  do
  {
    echo "a: $a<br>\n";
  }
  while  ($a <= 5);
  echo '</body></html>';
?>
```

Listagem 17. Usando o break

```
<?php
  echo '<html><body>';
  $a=1;
  do
  {
    echo "a: $a<br>\n";
    $a++;
    if ($a == 3)
    {
      break;
    }
  }
  while  ($a <= 5);
  echo '</body></html>';
?>
```

Listagem 18. Utilizando o Continue

```
<?php
  echo '<html><body>';
  $counter=0;
  $test=10;
  while ($counter < $test)
  {
    if ($counter %2)
    {
      $counter++;
      continue;
    }
    echo "Valor corrente de counter: $counter<br>";
    $counter++;
  }
  echo '</body></html>';
?>
```

Listagem 19. Usando o Continue para interromper mais de uma repetição

```
<?php
  echo '<html><body>';
  $counter=0;
  while ($counter == 0)
  {
    $counter++;
    $inner=0;
    while ($inner == 0)
    {
      $inner++;
      continue 2;
      echo "inner<br>";
    }
    echo "middle<br>";
  }
  echo "outer<br>";
  echo '</body></html>';
?>
```

Listagem 20. Usando o For

```
<?php
  echo '<html><body>';
  for($i = 0; $i < 5; $i++)
  {
    echo "i: $i<br>\n";
  }
  echo '</body></html>';
?>
```

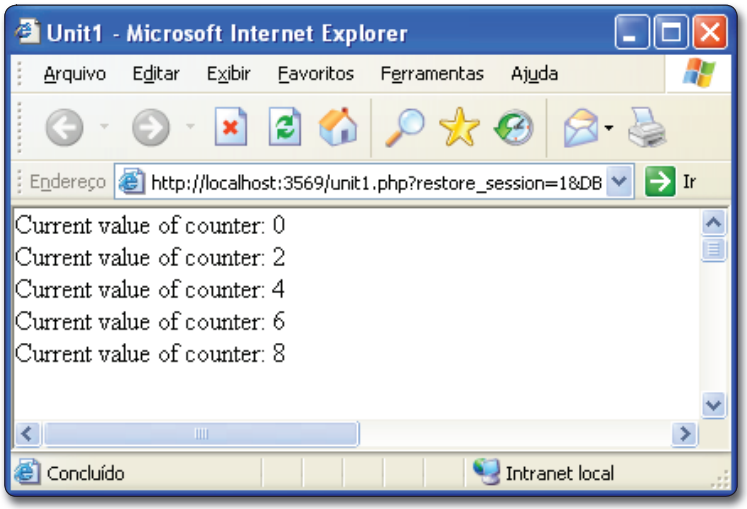


Figura 3. Exemplo do uso do Continue

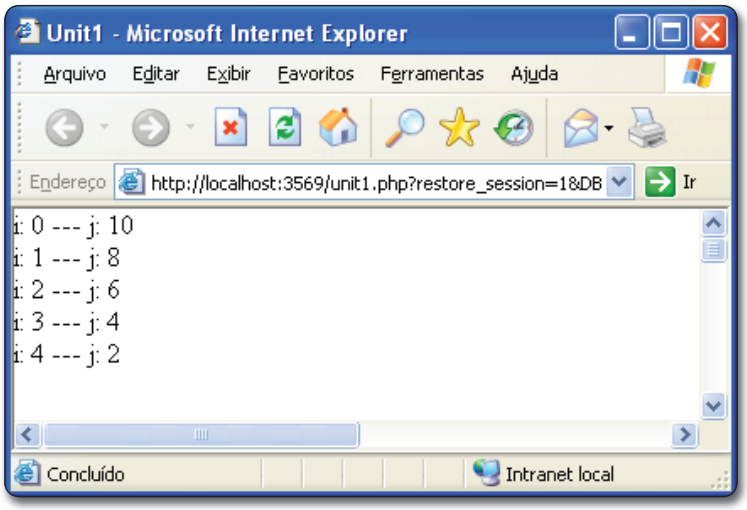


Figura 4. Usando o For no PHP

çará a verificar a condição novamente e se for ainda *true*, o bloco também será executado novamente. A saída do *script* é mostrada na **Figura 3**.

Uma linha é gerada para cada segundo número que a repetição processa. O *continue* também pode ser utilizado para interromper mais de uma repetição. Vejamos no exemplo da **Listagem 19**.

Podemos ver duas repetições *while*. Todas as repetições são executadas, porém, na repetição interna, chamamos *continue* com um parâmetro que diz ao PHP quantas repetições devem ser permitidas. Nesse caso dizemos ao PHP para executar duas repetições, portanto, o resultado estará fora do *break*. Nada mais será exibido na tela. Aplicações mais eficientes são construídas utilizando-se o *continue* com parâmetros, mas isso é também muito perigoso, pois o programador pode facilmente deixar de visualizar o fluxo lógico do código-fonte.

For

Se um bloco tiver que ser executado um número fixo de vezes, é recomendado utilizar *for* em lugar do *while*. Uma repetição *for* compõe-se de um valor inicial, uma condição que é verificada a cada vez que a repetição é processada e uma expressão que é executada depois que a repetição é processada. Vejamos um exemplo no código da **Listagem 20**.

Queremos exibir os números de 0 a 4 na tela. Para esse fim, um *for* foi implementado no *script*. A repetição começa com *$i* tendo o valor 0 e é executada enquanto *$i* for menor do que 5. O terceiro parâmetro diz ao PHP para incrementar *$i* a cada vez que a repetição for processada. A saída do *script* mostra o que é exibido quando o programa é iniciado utilizando um navegador Web:

```
i: 0
i: 1
i: 2
i: 3
i: 4
```

Às vezes, é necessário implementar repetições mais poderosas e mais complexas. Assim como em muitas outras linguagens ao estilo C ou C++, uma lista de condições pode ser utilizada em

combinação com *for*. Essa característica é muito importante porque permite que com pouco esforço, sejam implementadas funcionalidades complexas, como na **Listagem 21**.

Os três parâmetros compõem-se de uma lista de expressões utilizadas para processar certa variável. O texto gerado pelo programa mostra o que aconteceu às variáveis do programa (**Figura 4**).

A variável *$i*, foi incrementada de um, ao passo que *$j* foi decrementada de dois a cada vez que a repetição foi processada. A utilização da sintaxe que acabamos de ver permite que sejam tratadas múltiplas variáveis, utilizando apenas um *for*. A lista de operações utilizadas na repetição pode ser facilmente estendida para a complexidade desejada.

Até agora, aprendemos a utilizar *for* com uma lista de expressões. Contudo, em PHP, os programadores não são forçados a fornecer uma lista de pelo menos um parâmetro, pois é também possível deixar a lista vazia como mostrado no exemplo da **Listagem 22**.

O primeiro e o terceiro parâmetros do *for* estão vazios. O resultado da repetição é ainda uma lista de todos os números de 0 a 4, porém foi gerada de forma diferente. A variável *$i* é inicializada na linha anterior ao início da repetição e é incrementada após exibir a saída na tela. Há várias maneiras de resolver o problema, pois o PHP tem uma sintaxe comparativamente rica. Para aqueles que desejam escrever código *script* bem curto e eficiente, o exemplo a seguir é bastante interessante:

```
<?php
  echo '<html><body>';
  for ($i = 0; $i < 5; print "i: $i<br>", $i++);
    echo '</body></html>';
?>
```

A saída do *script* é ainda a mesma, mas utilizamos apenas uma linha para exibir os números na tela. Como podemos ver, o PHP permite que utilizemos qualquer espécie de expressões em um *for*, o que torna esse tipo de repetição extremamente flexível.

Switch

Para evitarmos escrever seqüências longas de cláusulas *if*, às vezes é possível utilizar cláusulas *case* e *switch* no seu lugar. As cláusulas *switch* e *case* muitas vezes são utilizadas para selecionar o trecho de código correto que deve ser executado, dentre uma lista de possibilidades. Vejamos no exemplo da **Listagem 23.**

Listagem 21. Repetições For mais complexas

```
<?php
  echo '<html><body>';
  for($i = 0, $j = 10; $i < 5, $j > 0;
    $i++, $j = $j - 2)
  {
    echo "i: $i --- j: $j<br>\n";
  }
  echo '</body></html>';
?>
```

Listagem 22. Deixando a lista vazia

```
<?php
  echo '<html><body>';
  $i=0;
  for (; $i < 5; )
  {
    echo "i: $i<br>\n";
    $i++;
  }
  echo '</body></html>';
?>
```

Listagem 23. Usando um switch

```
<?php
    $a=2;
     switch ($a)
    {
        case 1:
            echo "case 1: \$a é $a<br>";
            break;
        case 2:
            echo "case 2: \$a é $a<br>";
            break;
        case 3:
            echo "case 3: \$a é $a<br>";
            break;
    }
?>
```

Listagem 24. Usando um switch com default

```
<?php
    $a=9;
     switch ($a)
    {
        case 1:
            echo "case 1: \$a é $a<br>";
            break;
        case 2:
            echo "case 2: \$a é $a<br>";
            break;
        default:
            echo "o valor não foi encontrado<br>";
    }
?>
```

Em primeiro lugar, 2 é atribuído a $a. A cláusula *switch* verifica todos os casos e, se o valor correto for encontrado, o bloco será ativado e o PHP executará todo o código até que um comando *break* seja encontrado. Se executarmos o script, apenas uma linha será exibida:

```
case 2: $a é 2
```

Outra característica das cláusulas *switch* e *case*, é a capacidade de lidar com valores default. Se nenhum valor da lista casar, o PHP executará o ramo default. No exemplo da **Listagem 24**, estamos procurando 9, mas 9 não foi listado na cláusula *switch*.

O PHP executará o código default porque o 9 não foi encontrado na lista.

As construções *switch* e *case* são uma característica muito flexível e poderosa do PHP. É possível construir aplicações sofisticadas sem necessidade de utilizar as construções mais complexas if/else.

Conclusão

Vimos nesta primeira parte uma introdução à linguagem PHP, e também as construções primárias da linguagem, como variáveis, estruturas condicionais e de repetição. Até a próxima edição. ●

Artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

SEÇÃO

Delphi for PHP

Crie um site de publicação de notícias – Parte 2

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

Na primeira parte deste mini-curso (edição 88) criamos o banco de dados, a estrutura inicial da aplicação, com o Data Module e, montamos telas de manutenção de registros. Para este artigo o assunto abordado é a criação de páginas de pesquisas utilizando a tecnologia Ajax e uma página de consulta a dados.

Formulário de pesquisa

Para a criação da página de pesquisa precisamos de um novo formulário na aplicação. Crie o novo formulário através do menu *File>New>Form*. Como neste exemplo será criado a página de pesquisa de instituições altere a propriedade *Name* do formulário para "UPesInstituicoes" e salve o arquivo como "pesInstituicoes.php".

Adicione ao formulário um *Label* da paleta *Standard* e altere a propriedade *Caption* para "Digite um nome:". Ao lado coloque um *Edit*, também da paleta *Standard* e altere seu *Name* para "edtPesquisa". Ao lado do *Edit* adicione um *Button* a altere o *Caption* para "Pesquisar".

Teoricamente não precisaríamos adicionar componentes de acesso a dados nesse formulário, pois poderíamos utilizar os mesmos componentes que estão no Data Module. No entanto, acho um desperdício de desempenho trazer campos que não utilizaremos na exibição. Pense no caso de nossa pesquisa de instituições, onde bastaria apresentar os campos código e nome e não tem porque trazer todos os campos. Aliás, o banco agradece esse cuidado.

Então nesse formulário adicionaremos um *Query*, da paleta *Data Access*. Para ligar a propriedade *Database* é necessário antes acessar o menu *File>Use Unit*, escolher *dados.php* e pressionar OK. Com isso, poderemos definir a propriedade *Database* para *UDados.dbjornal1*. Para a propriedade *Name* informe "sqlInstituicoes" e na SQL digite o seguinte:

```
select id_instituicao, instituicao
from instituicoes
```

A instrução SQL parece estranha, pois, pelo conhecimento em Delphi, você deve estar se perguntando cadê o parâmetro e a condição para trazer determinados registros que estão sendo procurados?

O Delphi for PHP tem como característica trabalhar com a propriedade *Filter* (nossa conhecida de outras versões), que funciona adicionando uma cláusula condicional no SQL do tipo *where*. Então é uma preocupação a menos, basta utilizar o *Filter* que o Delphi for PHP se encarregará de fazer o *where*.

No mesmo sentido existe a propriedade *OrderField* que especifica a forma de ordenamento da instrução, adicionando uma cláusula *Order By*. Para o exemplo, informe "instituicao" nessa propriedade. Na verdade o *Query* do Delphi for PHP pode ser considerado uma evolução do *Query* do Delphi 7, pois facilita o controle da instrução SQL que será executada no banco de dados.

Agora adicione um *Datasource* ao formulário ligando sua propriedade *DataSet* para *sqlInstituicoes* e alterando seu *Name* para "dsInstituicoes". Para exibir o resultado da pesquisa poderíamos utilizar um *DBGrid*. Mas para esse exemplo será utilizado o *DBRepeater* pois a idéia é que, a cada registro resultado, seja feito um link para a página de manutenção de registros passando uma *QueryString*, que informará o código da instituição.

Pode-se considerar que o *DBRepeater* é parecido com uma banda *Detail* no *QuickReport* e todos os componentes que estarão dentro do mesmo serão repetidos para cada registro resultante da pesquisa. Então adicione um *DBRepeater* no formulário. Deveríamos ligar a propriedade *DataSource* mas não farei porque isso exige que a *Query* esteja aberta (*Active=True*) no carregamento da página, o que degradaria a performance da página, trazendo todos os registros sem tê-los solicitado. Portando, faremos isso via código, no momento da pesquisa.

Adicione dois *Labels* dentro do *DBRepeater* e faça a ligação da propriedade *DataSource* para *dsInstituicoes*. Para a propriedade *DataField* informe "id_instituicao" para o primeiro e "instituição" para o segundo. No primeiro *Label* altere a propriedade *Name* para "lblCodigo". Para finalizar, ajuste a altura do *DBRepeater* lembrando que será a altura impressa para cada registro. Veja o resultado da tela na **Figura 1**.

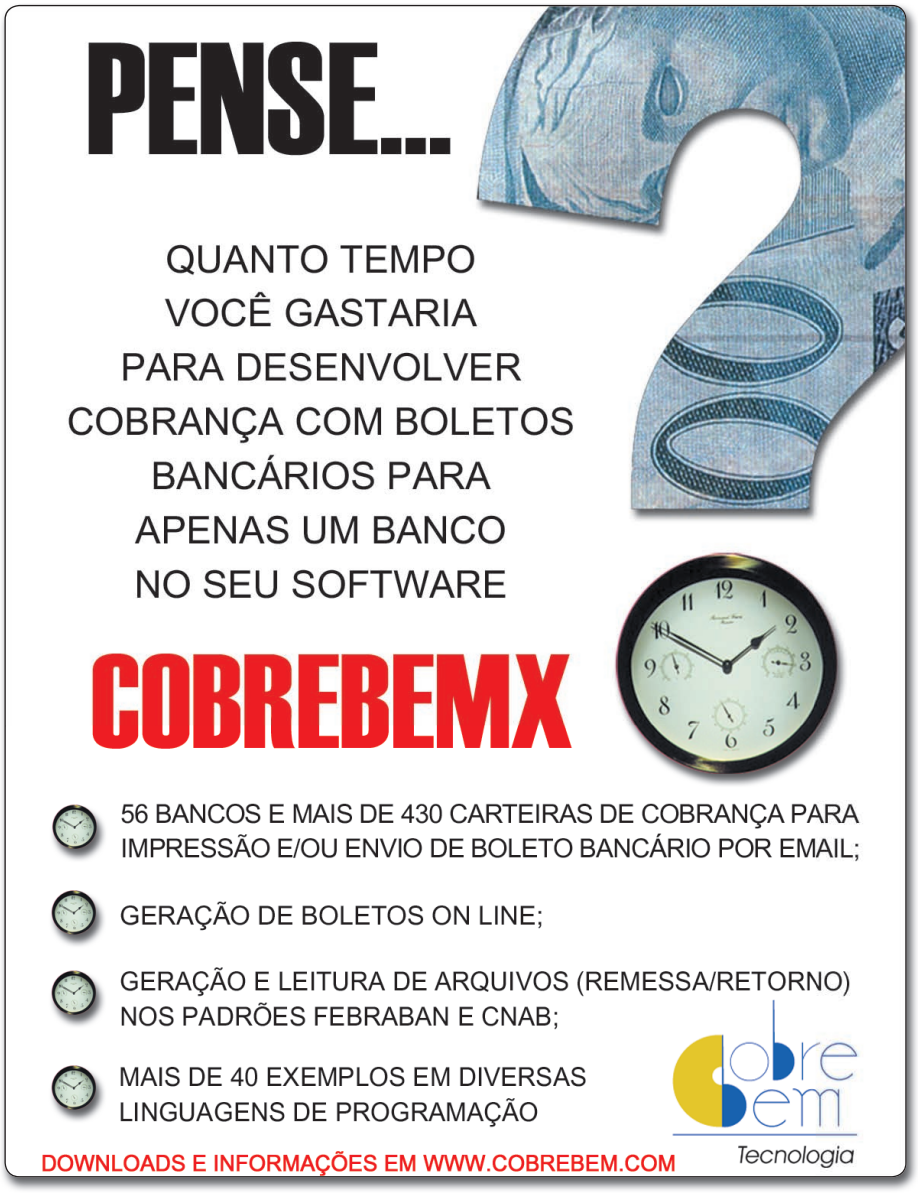
Codificando a pesquisa

Vamos ao código. Clique duas vezes sobre o *Pesquisar* e informe o código da **Listagem 1**.

Explicando o código anterior, temos um *Close* no *Query* para certificar que consultas anteriores no mesmo formulário estarão fechadas. Passamos para a propriedade *Filter* uma instrução do tipo *Like* (como). Para possibilitar que o usuário não se preocupe com letras maiúsculas ou minúsculas utilizamos a função *Upper* existente no MySQL, que deixa tudo em maiúsculo.

Buscamos o valor existente no *Edit* e acrescentamos um caractere coringa (%), que funciona determinando que parte do texto (anterior) deve ser encontrado e o restante tanto faz. Por exemplo: se informar "a%" serão exibidas todas as instituições que a letra inicial for "a". Para finalizar abrimos a consulta (*Open*) e informamos a propriedade *DataSource* do *DBRepeater*.

Nesse momento, podemos fazer os primeiros testes. Compile a aplicação e tente localizar alguns registros. Lembre-se que se não for informado nada no *Edit* e for



Listagem 1. Código do Pesquisar

```
$this->sqlInstituicoes->Close();
$this->sqlInstituicoes->Filter=
  'Upper(instituicao) like Upper("'.$this->edtPesquisa->Text.'%")';
$this->sqlInstituicoes->Open();
$this->DBRepeater1->DataSource=dsInstituicoes;
```

Listagem 2. Validando o Edit de pesquisa

```
msg="";
result=true;
if ((document.UPesInstituicoes.edtPesquisa) &&
  (document.UPesInstituicoes.edtPesquisa.value==""))
{
  result=false;
  msg+="-Preencha um critéio para a Pesquisa\n";
}
if (!result)
{
  alert(msg);
}
return(result);
```

Listagem 3. Código do evento OnBeforeShow da página instituicoes.php

```
//Pegar o valor do QueryString
$codigo=$this->input->id;
if (is_object($codigo))
{
  //Transformar a classe em variável local
  global $UDados;
  ..
  //Filtra pelo id
  $UDados->tbinstituicoes1->Filter="id_instituicao=".
    $codigo->asInteger();
  $UDados->tbinstituicoes1->Refresh();
}
```

Listagem 4. Evento JavaScript do Pesquisa

```
function Button1JSClick($sender, $params)
{
  echo $this->Button1->ajaxCall("Busca");
  ?>
  //Add your javascript code here
  msg="";
  result=true;
  if ((document.UPesInstituicoes.edtPesquisa) &&
    (document.UPesInstituicoes.edtPesquisa.value==""))
  {
    result=false;
    msg+="-Preencha um critéio para a Pesquisa\n";
  }
  if (!result)
  {
    alert(msg);
  }
  return(false);
  <?php
}
```

Listagem 5. Código da função Busca

```
function Busca($sender, $params)
{
  $this->sqlInstituicoes->Close();
  $this->sqlInstituicoes->Filter=
    'Upper(instituicao) like Upper("'.$this->
    edtPesquisa->Text.'%")';
  $this->sqlInstituicoes->Open();
  $this->DBRepeater1->DataSource=dsInstituicoes;
}
```

pressionado o *Pesquisar*, todos os registros serão exibidos.

Se você deseja exigir do usuário um critério mínimo para pesquisa, podemos fazer a validação do *Edit*. Para isso, no evento *OnClick* do botão, na aba *Javascript*, abaixo do comentário, coloque o código da **Listagem 2**. Veja na **Figura 2** a aplicação sendo executada.

Para completar a página de pesquisa vamos fazer um link para a página de manutenção de instituições. Vamos fazer o link no *lblCodigo* que está ligado ao campo *id_instituicao*. Selecione esse componente e codifique seu evento *OnBeforeShow* com o código a seguir:

```
global $UDados;
$this->lblCodigo->Link="instituicoes.
php?id=".$this->
  sqlInstituicoes->id_instituicao;
```

O código anterior é muito simples. Utiliza-se a propriedade *Link* para atribuir um link ao componente. Como esse evento é disparado a cada interação do *DBRepeater* será adicionado ao link um campo de consulta com o id corrente. Com isso tem-se a chamada à página de manutenção (*instituicoes.php*) passando o *id* da instituição que será exibida. Veja na **Figura 3** a mudança da tela de pesquisa.

Para que a página de manutenção abra o registro selecionado na pesquisa precisa-se codificar o evento *BeforeShow* da *instituicoes.php* conforme o código da **Listagem 3**.

Para buscar um campo de consulta utilizamos o *$this->input->nome* que tem o mesmo funcionamento de *$_GET['nome']* que a forma padrão do PHP para buscar campos existentes no link. Para verificar se existe um valor no link verificamos se a variável é um objeto, o que, em outras palavras significa que existe e possui um valor.

Como a variável torna-se um objeto é necessário utilizar o *asInteger* para transformar em um número inteiro. Teste novamente a aplicação fazendo uma pesquisa e clicando no link para verificar se o registro escolhido será apresentado.

Usando Ajax

Utilizar a tecnologia Ajax nas aplicações Delphi for PHP é muito fácil. Exemplificarei modificando nossa consulta para buscar os dados sem dar *refresh* na página. Inicialmente é necessário alterar

a propriedade *UseAjax*, do *pesInstituicoes.php* para *True*.

Isso avisará que podemos ter chamadas assíncronas no servidor. Continuando, é necessário retirar o código do *OnClick* do *Pesquisar*. Como sugestão, sugiro que guarde o código em outro lugar, pois ele será utilizado em outro evento.

> **Nota:** Depois de retirar o código de um evento, é necessário salvar a unit, para que a estrutura do evento seja excluída. Em alguns casos, de forma manual precisamos apagar a chamada ao evento no *Object Inspector*. Se isso não for realizado será apresentado um erro na execução.

Para chamar uma execução utilizando Ajax, utilizamos o *OnClick* da aba *Javascript*, apresentada no *Object Inspector*. Selecione o *Pesquisar*, procure por essa aba de evento e codifique-o conforme o seguinte código:

```
echo $this->Button1->ajaxCall("Busca");
```

O *echo* é um comando utilizado para escrever valores. O *ajaxCall* existe em todos os componentes para possibilitar a chamada a uma função utilizando a tecnologia Ajax. No exemplo estamos chamando uma função que possui o nome *Busca*.

Ainda precisa-se impedir que o *PostBack* da página seja executado. Para tanto coloque *return(false);* no final do código que colocamos para validar o critério de busca, abaixo do comentário *//Add your*

Listagem 6. Evento OnChange do ListBox

```
function ListBox1JSChange($sender, $params)
{
  ?>
  //Add your javascript code here
  params=document.UConsulta.ListBox1.options[
    document.UConsulta.ListBox1.value].text;
  <?php
  echo $this->ListBox1->ajaxCall("BuscarTabela");
}
```

Listagem 7. Código da função BuscarTabela

```
function BuscarTabela($sender,$params)
{
  $this->Table1->Close();
  $this->Table1->TableName=$params;
  $this->Table1->Open();
}
```

Figura 1. Tela de pesquisa, usando o DBRepeater

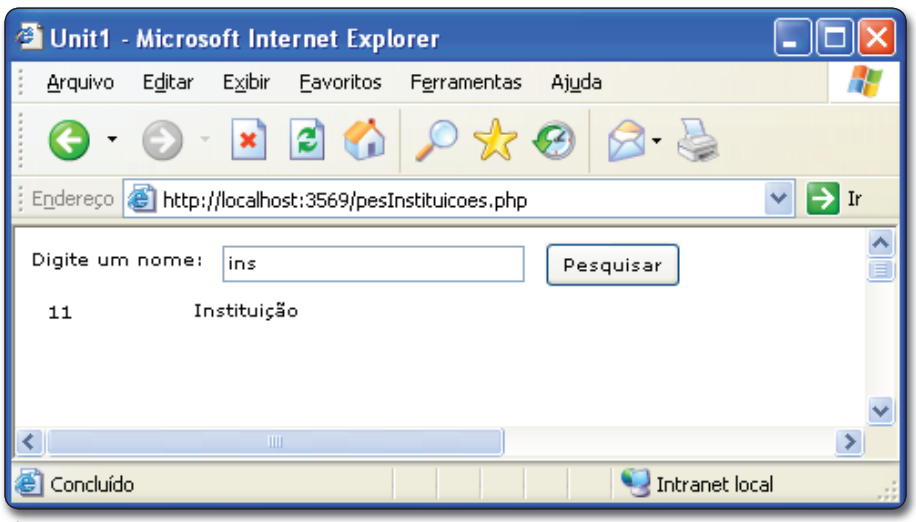


Figura 2. Página de pesquisa em execução

javascript code here ainda no mesmo evento. Veja no código da **Listagem 4**, como fica todo o evento.

Por último, é necessário criar a função *Busca* o que pode ser feito após o fechamento do evento *Button1JSClick*. Codifique a função conforme a **Listagem 5**. Se você ainda possui o código que retiramos anteriormente, utilize-o pois não muda nada. Apenas criamos uma função fora de um evento para ser chamada usando Ajax.

Pronto agora é só testar e ver a pesquisa funcionando com Ajax. Aplique seus conhecimentos adquiridos até agora e crie a pesquisa para usuários e notícias. Se sentir dificuldade baixe o código-fonte do artigo no site da revista.

Página de consulta

Essa página de consulta a dados servirá para exibir os dados de uma tabela escolhida em um *ListBox*. Crie um novo formulário no menu *File>New>Form* e defina sua propriedade *Name* para "UConsulta", salve-o como "consulta.php" e defina a propriedade *UseAjax* como *True*.

Adicione um *ListBox* e um *DBGrid* ao formulário. Adicione também um *Table* e um *DataSource*. Ligue o *DataSource* no *Table*, através da propriedade *DataSet*. Ligue o *Table* no *DataBase* que está na unit *Dados*.

Na propriedade *TableName* do *Table* informe "instituicoes" e em *Active* escolha *True*. Ligue o *DBGrid* ao *DataSource* através da propriedade *DataSource*. Agora você deve codificar o evento *OnBeforeShow* do formulário com o código exibido a seguir:

```
global $UDados;
$tabelas = $UDados->dbjornal1->tables();
$this->ListBox1->Items=$tabelas;
```

O código é responsável por buscar o nome das tabelas existentes no banco de dados e adicioná-los ao *ListBox*. Clique no *ListBox* e acesse o *Object Inspector*, na aba *Javascript* acesse o evento *OnChange* para codificá-lo como mostra a **Listagem 6**.

Na listagem anterior utilizamos um código *JavaScript* para pegar o valor selecionado na lista e passar para a variável *params* que poderá ser acessada na execução da função. Depois chamamos a execução da função *BuscarTabela* que será criada. Abaixo do último evento adicione a função apresentada na **Listagem 7**.

A função *BuscarTabela* fecha a tabela, recebe o nome da tabela que deverá ser apresentada através da variável *params* e a abre exibindo os dados. Faça o teste, compilando sua aplicação e acessando as tabelas existentes no banco de dados. Lembre-se que o *DBGrid* permite a edição dos dados, salvando-os utilizando a tecnologia Ajax. Veja na **Figura 4** como deve ficar essa página.

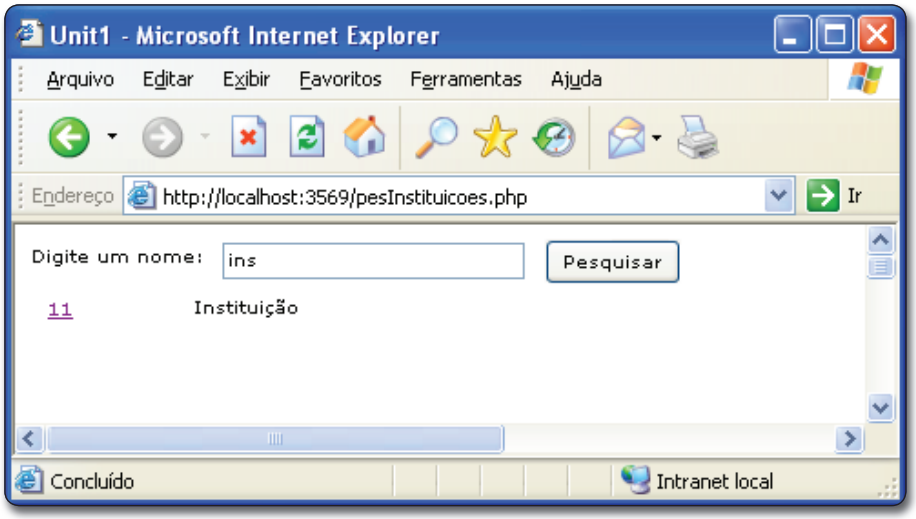


Figura 3. Link de redirecionamento para a página de cadastro

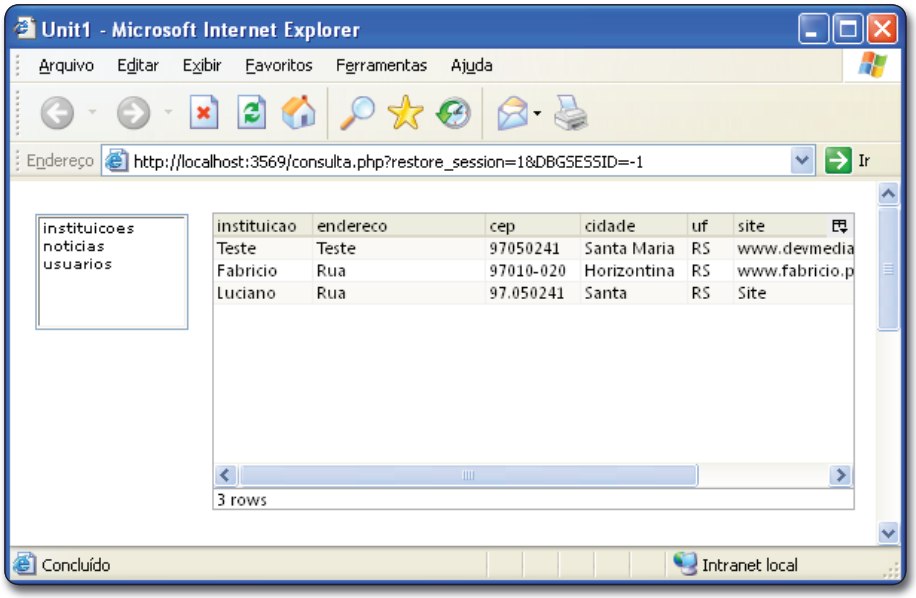


Figura 4. Tela de consulta genérica

Conclusão

Neste artigo vimos como criar páginas de pesquisa utilizando o *Like* e também a criação de uma página de consulta genérica onde podemos escolher qualquer tabela no banco de dados. Como tenho comentado nos artigos de Delphi for PHP, é extremamente fácil aprender e trabalhar com essa ferramenta. Então, tente você também! ●

> **Nota:** Enquanto escrevia esse artigo a CodeGear liberou o Update 2 do Delphi for PHP. É recomendado que a atualização seja feita com urgência. Maiores informações: *dn.codegear.com/article/36864*.

Um bom software também precisa de um bom layout!

Kameha
webdesign

Vídeo-aulas especializadas em Design Digital.

- Aprenda através de vídeo-aulas 100% práticas, tudo que as ferramentas de design podem lhe oferecer. Desde o básico ao mais avançado recurso.
- Assine agora mesmo e receba:
  + de 50 vídeo-aulas de acesso imediato
  + de 10 vídeos inéditos por mês.

Por apenas 6x de **R$ 13,61**

Estas são algumas das ferramentas que já estão disponíveis para você:

PhotoShop	Corel Draw
Flash	After Effects
DreamWeaver	e muito mais...

Acesse e assine já!

www.kameha.com.br

Um produto da DevMedia GROUP

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDelphi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435**